OPTIMIZING THE STRUCTURE OF DIFFUSION NETWORKS: THEORY AND ALGORITHMS

A Thesis
Presented to
The Academic Faculty

by

Elias B. Khalil

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Computer Science

Georgia Institute of Technology
May 2014

OPTIMIZING THE STRUCTURE OF DIFFUSION NETWORKS: THEORY AND ALGORITHMS

Approved by:

Professor Le Song, Advisor
School of Computational Science & Engineering
Georgia Institute of Technology

Professor Bistra Dilkina
School of Computational Science & Engineering
Georgia Institute of Technology

Professor Duen Horng (Polo) Chau
School of Computational Science & Engineering
Georgia Institute of Technology

Date Approved: 27 March 2014

To my parents.

ACKNOWLEDGEMENTS

I would like to thank Professor Le Song for proposing this topic early in 2013. Professor Song's guidance has pushed me beyond the limits of my knowledge, and introduced me to the world of quality computing research. I also thank Professor Bistra Dilkina for dedicating time to helping me critically examine my work and improve upon it, as well as to expressing my ideas in a clear and concise way. I also greatly appreciate Professor Polo Chau's support and guidance since I have come to Georgia Tech in 2012. Since my first week at Tech, Professor Chau has agreed to give me the chance to engage in his research projects, and it is for that very reason that I have developed interest in the area of network analysis and data mining.

I thank my parents, Boutros and Marlene, for their love and endless support; they are the sole reason I am here today. Last but not least, I thank my family and friends back home, who have showed nothing but support all throughout these two years, despite being thousands of kilometers away.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

How can we optimize the topology of a networked system to make it resilient to flus or malware, or also conducive to the spread of information and multimedia? Previous work on information diffusion has focused on modeling the diffusion dynamics and selecting nodes to maximize influence (e.g. by offering a product for free as part of a marketing campaign). Only a paucity of recent studies have attempted to address the network modification problems, where the goal is to either facilitate desirable spreads or curtail undesirable ones by *adding or deleting* a small subset of network nodes or edges, as opposed to simply *seeding* existing nodes.

In this thesis, we focus on the widely studied *linear threshold diffusion model*, and prove, for the first time, that the network modification problems under this model have *supermodular* objective functions. This surprising property facilitates computational advancements on multiple fronts: it allows us to design efficient data structures and scalable algorithms with provable approximation guarantees, despite the hardness of the problems in question. Both the time and space complexities of our algorithms are *linear* in the size of the network, which allows us to experiment with networks of millions of nodes and edges. We show that our algorithms outperform an array of heuristics in terms of the effectiveness in controlling diffusion processes, often beating the next best by a significant margin.

CHAPTER I

INTRODUCTION

The diffusion of physical, conceptual or digital substances over networks has been studied in many domains such as epidemiology [19], social media [23, 29], and computer [44] and mobile [34] networks. Such substances may be diseases, rumors, malware, etc., depending on the application setting.

Previous studies have resulted in an array of models aiming at capturing the diffusion dynamics, among which the most well-known are the linear threshold (LT) model, the independent cascade (IC) model, and the Susceptible Infected Recovered (SIR) model. These models are stochastic in nature, with contagion being transmitted from a node to its neighbors with some probability, or also dying out with some probability (as in SIR).

Another line of research has concentrated on how one can select a small set of source nodes whose initial adoption of a given contagion would trigger maximal spread in the network. The problem of *source node selection problem for influence maximization* was first proposed in the context of viral marketing by Richardson and Domingos in [12]. Following that, Kempe et al. formulated the same problem as a discrete optimization problem. In this setting, given a network and a corresponding diffusion model, the goal is to select a set of k source nodes whose initial adoption of a given substance would trigger maximal spread in the network. While this problem is NP-hard in general, Kempe et al. [23] show that the objective function of the node selection problem is *submodular* for the LT and IC models, an immediate consequence of which is that a simple greedy algorithm provides solutions within $(1 - 1/e)$ approximation

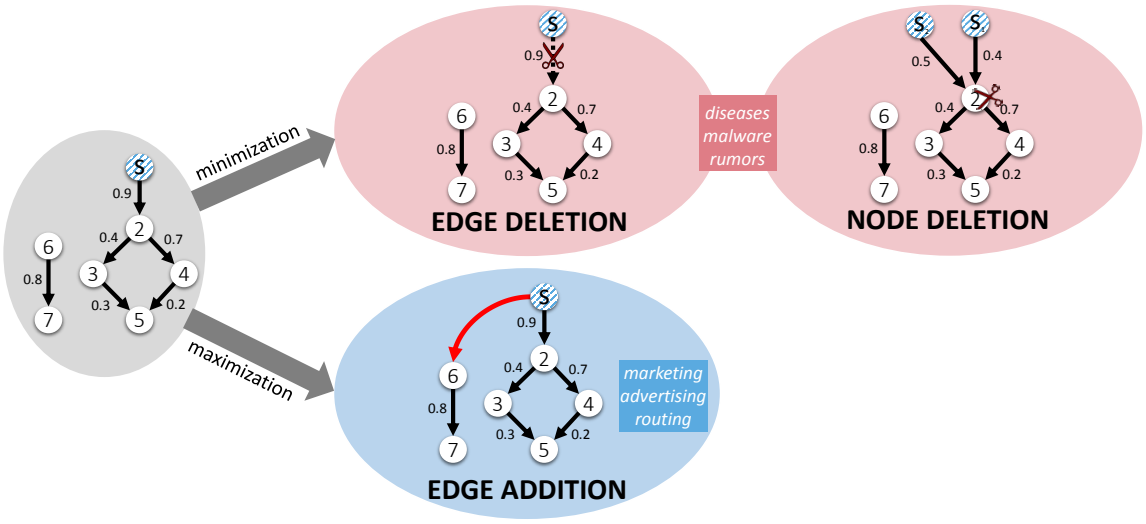


Figure 1: The two main problems we address in this paper: diffusion minimzation and maximization in a network under the LT diffusion model. Source node is dashed in blue, deleted edge is dashed, added edge is red.

factor. This result also generated a whole line of research on using submodular optimization for source node selection in various related information diffusion problems, and on designing efficient algorithms for tackling large-scale networks [8, 7, 9, 10, 13].

In contrast to these previous works where *the diffusion networks remain unchanged*, we are interested in problems of *modifying the topology of a diffusion network* to either facilitate the spread of desirable substances, or curtail the spread of undesirable ones. One can consider deleting edges or nodes to minimize a possible undesirable spread, such as that of a virus, disease or rumor. For instance, in disease control, authorities may consider disallowing travel between certain pairs of cities to curb the spread of a flu epidemic. Similarly, one can consider adding edges or nodes to facilitate the spread of, for example, information or endangered species. As an example, social media websites can recommend to users additional information outlets to follow to increase the spread of ideas and memes. The network modification setting is particularly relevant when the agent optimizing the topology does not have control over the sources of the substance that is spreading, but is able to change some subset of the edges or nodes that he has access to. Despite the broad practical relevance of

these network modification problems, existing results on network topology optimization for diffusion processes are very limited, and lack in either proper formulations, formal optimality guarantees or in algorithmic efficiency.

1.1 Thesis Statement

In this thesis, we will address network topology optimization for diffusion under the *linear threshold model*. We will focus on three network modification problems involving the deletion of edges or nodes for minimizing spread, and the addition of edges for maximizing spread [1] :

> **Edge deletion problem:** Given a set of X source nodes, find a set of k edges to remove such that the spread of a certain substance is minimized.
>
> **Node deletion problem:** Given a set of X source nodes, find a set of k nodes to remove such that the spread of a certain substance is minimized.
>
> **Edge addition problem:** Given a set of X source nodes, find a set of k edges to add such that the spread of a certain substance is maximized.

Our algorithms 1) produce solutions with approximation guarantees, 2) are scalable to large networks and 3) experimentally perform significantly better than heuristics.

1.2 Our Approach to Diffusion-aware Network Optimization

1.2.1 Optimization with guarantees

We will first prove a surprising result, namely that the objective function in all three problems is *supermodular*, a property that has positive algorithmic implications. In particular, minimizing a supermodular function under cardinality constraints, although typically an NP-hard problem, admits a greedy algorithm with approximation

[1] One can also consider the analogous node addition problem, to which our theoretical and algorithmic results for the three problems can easily be extended.

guarantees [33]. Similarly, cardinality-constrained supermodular maximization has recently been shown to admit a simple modular approximation scheme [22, 21]. Our finding, combined with these combinatorial optimization results allows, for the first time, the design of efficient diffusion-aware algorithms with approximation guarantees for the network modification problems under the LT model.

1.2.2 Scalable algorithm design

We address several challenges at the core of designing the two general supermodular approximation algorithms. Our goal is to obtain efficient algorithms that are scalable and practical with respect to large-scale network datasets. However, directly implementing the supermodular optimization algorithms is impractical, since evaluating the objective function given a set of source nodes is #-P hard in general [9]. We exploit the correspondence between the LT model and the "live-edge graph" construction [23], and estimate the objective function using a sample of random live-edge graphs. Still, a naive application of the supermodular approximation schemes to the sample of random live-edge graphs will result in runtime quadratic in the network size, which cannot scale to modern problems with millions of nodes and edges. To tackle this issue, we design two data structures, the *descendant-counting trees* for the edge and node deletion problems, and the *neighbor-counting graphs* for the edge addition problem, in order to support fast approximate evaluation of the objective function. These data structures can be *constructed in time linear in the network size*, and *queried in constant time*, allowing us to scale the supermodular optimization algorithms to networks with millions of nodes and edges.

1.2.3 Experimental evaluation

Finally, we evaluate our algorithms on both synthetic and real-world diffusion networks and compare the quality of the solutions to alternative approaches, based on optimizing structural properties of the networks. Our algorithms outperform all other

heuristics across all experiments, leading to as large as 10-20% additional efficacy for edge deletion, and up to 100% for edge addition, compared to other approaches that rely on the structural properties of the network (weights, shortest paths, eigenvalues, degrees, etc), not making use of the probabilistic diffusion model. In terms of running time, our algorithms scale linearly to large networks with millions of nodes and edges.

1.3 Contributions

Our contributions can be summarized as follows:

1. We propose a set of diffusion-aware network optimization problems.

2. We present a series of proofs for the supermodularity of the various objective functions proposed.

3. We design fast approximate algorithms using efficient data structures and randomized estimation techniques.

4. We demonstrate experimentally the scalability of the algorithms to million-scale networks, and their larger effectiveness as compared to heuristics.

1.4 Organization of the thesis

We will first survey some related work in a set of relevant research contexts (chapter 2). Then, we present background on the linear threshold (LT) diffusion model and its correspondance to the "live-edge graph" process (chapter 3). We then analyze the space of live-edge graphs under the LT model, and extract some properties of this space in chapter 4. Next, we will provide the formal definitions of the network modification problems, and show that their objective functions are supermodular under the LT model (chapter 5). Then, we will design efficient data structures and algorithms to solve the supermodular optimization (chapter 6). Finally, we present our experimental setting and corresponding results (chapter 7), then conclude.

CHAPTER II

RELATED WORK

In this section, we survey related work across different research areas.

Graph Modification. The theoretical computer science literature addresses many problems related to graph modification. Often, the main question is that of finding the minimum number of nodes or edges to delete from a graph, such that the resulting subgraph satisfies a given combinatorial property. For instance, Yannakakis (and indepedently Rose and Tarjan) shows that such node and edge deletion problems are NP-Complete for various combinatorial properties such as acyclicity of a given length, transitivity, line-invertibility, etc [45, 32, 46, 37]. While most results for graph modification problems are negative, a subset of these problems allow for fixed-parameter tractable solution, when the given property satisfies some conditions, as demonstrated by Cai in [6].

In a rather different optimization setting, researchers have considered the problem of adding edges (from a set of candidates edges) to a graph so as to maximize its *algebraic connectivity*, or equivalently the second smallest eigenvalue of the graph Laplacian [17].

Our work differs from this rich literature in that we are optimizing the network topology relative to a dynamic process rather than combinatorial or spectral properties. Nevertheless, we suspect that our problems are also as hard as the class we just described.

Information Diffusion. A much more closely related area of research pertains to the study of information diffusion mechanisms and models. Everett Rogers popularized this field early in 1962 under the name of "diffusion of innovations" [36], where

a new theory of how ideas and technology spread through cultures was presented. In this seminal piece, Rogers synthesized results from over 500 previous studies on diffusion. The result was a theory of diffusion that emphasized the importance of social ties in the spreading process, and an analysis of the conditions under which a certain idea is capable of spreading and self-sustaining in society, among other things.

Since then, considerable amounts of web data have been made available through technological advancement, allowing for the study of diffusion in the context of the web and its social networks. For instance, an empirical study is carried out in [1] to quantify the influence of various Twitter users on the diffusion of information through tweets. Similarly, Leskovec et al. examine cascades of news among blog websites and analyze predominant diffusion patterns and methods for detecting news early on in their cycle [31, 27], while the authors of [28] look at the propagation of recommendations in a purchasing network.

We differ from this line of research in that we are not concerned with empirical analysis of diffusion phenomena, but rather consider the Linear Threshold model and attempt to perform some optimization tasks based on that.

Diffusion Optimization. Most related to this thesis are the papers we now briefly present, and that broadly deal with optimizing the structure of a diffusion network. On one hand, methods have been designed to optimize surrogates for diffusion processes [41, 38, 16]. Instead of maximizing/minimizing the spread of substances directly, these methods typically optimize a static property of the network, such as eigenvalues of the adjacency matrix or edge betweenness centrality, in the hope of optimizing diffusion. For instance, Tong et al. consider the edge deletion (addition) problems [41] and the node deletion problem [42] under the SIR model by approximately minimizing (maximizing) the eigenvalue of the adjacency matrix. Schneider et al. [38] proposed "betweenness centrality" as a heuristic for immunizing nodes or removing edges under the SIR model, while "degree centrality" was adopted in

[16] to delete nodes in order to protect against virus propagation in email networks . However, these surrogate-based methods often either: (1) do not have formal approximation guarantees, since they do not take into account diffusion processes explicitly, or (2) are based on the SIR suite of models, which are more particular to epidemiology. Among the works that directly address the diffusion processes, Sheldon et al. [39] study the problem of node addition to maximize spread under the IC model, and provide a counter-example showing that the objective function is *not* submodular. Thus, they resort to a principled but expensive approach based on sample average approximation and mixed integer programming, which provides provable optimality guarantees but cannot scale to large networks. Most existing approaches to solve diffusion network modification problems rely on heuristic algorithms without approximation guarantees. For instance, under the IC model, Bogunovic [2] addresses the node deletion problem. Kimura et al. apply the greedy algorithm used by Kempe et al. [23] for source node selection to the edge deletion problem under the IC model [25] and under the LT model [24], without analyzing the submodularity of the objective function or providing any formal guarantees. More recently, Kuhlman et al. [26] proposed heuristic algorithms for edge removal under a simple deterministic variant of the LT model. In contrast, we provide *fast* and *principled* algorithms for all three problems by exploiting supermodularity of the objectives under the LT model, and designing data structures and randomized subroutines, yielding linear-time complexity (up to logarithmic factors).

Supermodularity. Supermodularity is a mathematical property of set functions that we will exploit in this thesis in order to obtain reasonable approximations to our problems. Supermodular functions exhibit the property of "increasing differences",

which we will explain in Chapter 5. However, the mathematics and optimization literature mostly focuses on "submodular" functions [1], hence we will use the submodular optimization terminology, although methods transfer to supermodular functions. A celebrated result is that of Nemhauser et al. [33], showing that although maximizing a monotone submodular function (or, as in our case, minimizing a supermodular function) subject to cardinality constraints is NP-Hard, a greedy algorithm can provide a constant-factor approximation of around $(1 - 1/e) \simeq 63\%$. Since, multiple hardness and approximation results have been derived for various related submodular maximization problems, such as for knapsack constraints [40] or non-monotone submodular functions [15].

As for submodular minimization (or supermodular maximization), it is known that the unconstrained version of the problem is solvable in (higher-order) polynomial time using various methods, as surveyed in [20]. However, for the constrained submodular minimization problem, it is suspected that it falls within a different class of problems that are harder to solve. Fortunately, the recent work of Iyer et al. [22, 21] has resulted in simple approximation schemes for constrained submodular minimization, which we will exploit in order to solve our edge addition problem.

[1] If f is a supermodular function, then $-f$ is a submodular function

CHAPTER III

LINEAR THRESHOLD MODEL

In this chapter, we will provide background on the linear threshold (LT) model for diffusion processes [23], which will be the center of this study.

3.1 Cascade Generative Process

Underlying the LT model is a weighted directed graph $G = (V, E, w)$, called the *influence graph*, where V is a set of n nodes and E is a set of m directed edges, and $w : V \times V \to [0, 1]$ is a weight function. For edges $(u, v) \notin E$ we ignore the value of $w(u, v)$. We further require that $\sum_{u:(u,v)\in E} w(u, v) \leq 1$ for each node v. Starting from a source node (or an initially activated node) $S_0 = \{a\}$, a *cascade* then proceeds in discrete timesteps $t = 0, 1, 2, \ldots$ as follows:

1. at $t = 0$, every node v first independently selects a threshold θ_v uniformly at random in the range $[0, 1]$, reflecting the uncertainty regarding users' true thresholds;

2. subsequently, an inactive node v becomes activated at time $t + 1$ if

$$\sum_{u:u\in S_t,(u,v)\in E} w(u, v) \geq \theta_v$$

   where S_t is the set of nodes activated up to time t;

3. finally, the process terminates if no more activations are possible.

This process is illustrated in Figure 2.

Given an influence graph $G = (V, E, w)$, the *influence* function $\sigma(a, G)$ of a source node $a \in V$ is defined as the expected number of active nodes at the end of the

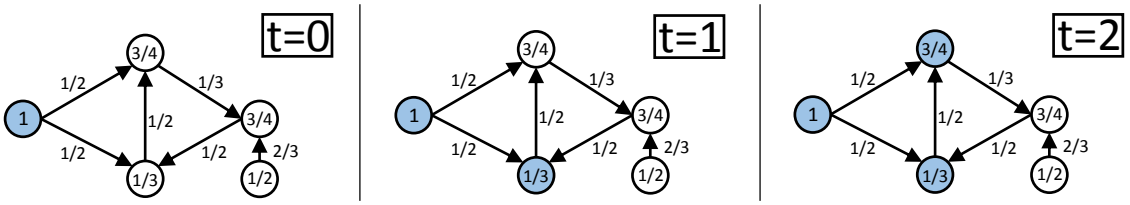


Figure 2: Illustration of the Linear Threshold diffusion process. Activated nodes are marked in blue, node thresholds are placed inside the nodes' circles, and edge weights are placed next to edges.

diffusion process, $\sigma(a, G) = \mathbb{E}[|S_\infty|]$, where the expectation is taken with respect to the randomness of the node thresholds θ_v.

3.2 Live-Edge Graph Representation

Kempe et al. [23] show that the influence function can be computed in an alternative way using what is referred to as "live-edge graphs", a construction that is more amenable to mathematical analysis. More specifically, a random live-edge graph X is generated as follows:

> Independently for each node $v \in V$, at most one of its incoming edges is selected with probability $w(u, v)$, and no edge is selected with probability $1 - \sum_{u:(u,v)\in E} w(u, v)$.

Note that the set of nodes of X is equal to V, the set of "live" (or sampled) edges E_X of X is a subset of E, *i.e.*, $E_X \subseteq E$, and these edges are unweighted. Clearly, each vertex has at most one incoming edge. Then, the influence function can be alternatively computed as:

$$\sigma(a, G) = \sum_{X \in \mathcal{X}_G} \Pr[X|G] \cdot r(a, X)$$

where $\mathcal{X}_G$ is the space of all possible live-edge graphs based on G, $\Pr[X|G]$ is the probability of sampling a particular live-edge graph X, and $r(a, X)$ is the set of all reachable nodes in X from source a.

If we define the function

$$p(v, X, G) := \begin{cases} w(u, v), & \text{if } \exists u : (u, v) \in E_X \\ 1 - \sum_{u:(u,v)\in E} w(u, v), & \text{otherwise} \end{cases}$$

which is the probability of the configuration of incoming edges for node v in X, then the probability of a particular live-edge graph X is:

$$\Pr[X|G] = \prod_{v \in V} p(v, X, G). \tag{1}$$

CHAPTER IV

SPACE OF LIVE-EDGE GRAPHS

In this section, we will prove four properties related to the space of live edge graphs which will form the basis of our later proofs of supermodularity in chapter 5. Our analysis will deal with the case of edge deletion, the results of which will be extended to the edge addition and node deletion cases in the next section. More specifically, we are concerned with

- a subset S of the edge set E in the original influence graph G, *i.e.*, $S \subseteq E$;
- and two distinct edges $e = (u, v) \in E \setminus S$ and $g = (u', v') \in E \setminus S$ outside S where v may or may not be equal to v'.

We will denote the modification of an influence graph G by deleting a set of edges S by:

$$G \setminus S := (V, E \setminus S, w).$$

Deleting a set S from E will result in a new influence graph $G \setminus S$ which will generate a new space of live-edge graphs, $\mathcal{X}_{G\setminus S}$, and the associated live-edge graph probabilities, $\Pr[X|G \setminus S]$. Furthermore, we will divide the space $\mathcal{X}_{G\setminus S}$, according to the edge e, into three disjoint partitions (see Figure 3(left)):

- $\mathcal{X}^{e}_{G\setminus S}$, the set of live-edge graphs where incoming edge $e = (u, v)$ is selected for node v;
- $\mathcal{X}^{\bar{e}}_{G\setminus S}$, the set of live-edge graphs where a different incoming edge $\bar{e} = (y, v)$ is selected for node v;
- $\mathcal{X}^{\emptyset}_{G\setminus S}$, the set of live-edge graphs where no incoming edge is selected for v.

Note that the probabilities of a live-edge graph X common to both $\mathcal{X}_G$ and $\mathcal{X}_{G\setminus\{e\}}$ may or may not change. Specifically, if node v has another incoming edge $g = (u', v) \neq e = (u, v)$ in X, then

$$\Pr[X|G] - \Pr[X|G \setminus \{e\}] = 0.$$

Otherwise, if node v has no incoming edge in X, then

$$\Pr[X|G] - \Pr[X|G \setminus \{e\}] = -w(u, v) \prod_{v' \neq v} p(v', X, G) \tag{2}$$

since all terms in Equation (1) for $\Pr[X|G]$ and $\Pr[X|G \setminus \{e\}]$ concerning nodes $v' \neq v$ are exactly the same, and

$$\begin{aligned} p(v, X, G) &= 1 - \sum_{u':(u',v)\in E\setminus\{e\}} w(u', v) - w(u, v) \\ &= p(v, X, G \setminus \{e\}) - w(u, v). \end{aligned}$$

We establish the following four relations between the spaces of live-edge graphs (see Figure 3 for illustrations).

4.1 Within Space Mapping

Our first result establishes a one-to-one mapping between the elements in partitions $\mathcal{X}^e_{G\setminus S}$ and $\mathcal{X}^\emptyset_{G\setminus S}$.

Proposition 1 *For every live-edge graph $X \in \mathcal{X}^\emptyset_{G\setminus S}$, there exists a corresponding live-edge graph $\widetilde{X} \in \mathcal{X}^e_{G\setminus S}$, and vice versa. If $X = (V, E_X)$, then $\widetilde{X} = (V, E_X \cup \{e\})$.*

Proof Since the edge $e \notin S$, we can always find two live-edge graphs within $\mathcal{X}_{G\setminus S}$ which differ by the edge e. The first live-edge graph X does not contain e, and the second live-edge graph X' contains the additional edge e. ■

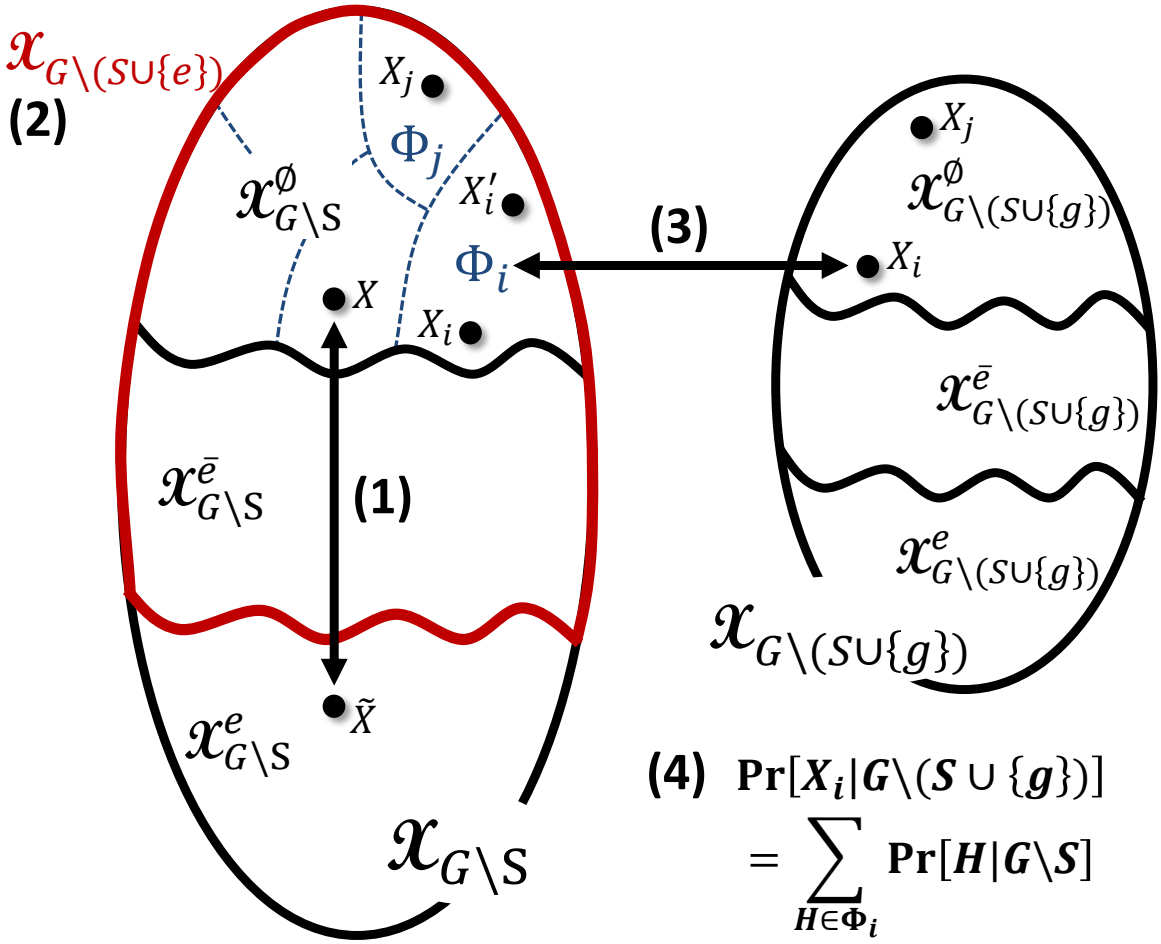


Figure 3: Four properties of the space of live-edge graphs: (1) within space mapping, (2) space inclusion, (3) across space mapping, and (4) across space probability mapping.

4.2 Space Inclusion

Our second result relates partitions $\mathcal{X}^{\bar{e}}_{G\setminus S}$ and $\mathcal{X}^{\emptyset}_{G\setminus S}$ of the space $\mathcal{X}_{G\setminus S}$ to the space $\mathcal{X}_{G\setminus(S\cup\{e\})}$. We note that this second space of live-edge graphs, $\mathcal{X}_{G\setminus(S\cup\{e\})}$, is generated from the influence graph $G\setminus(S\cup\{e\})$ with an additional edge e deleted from $G\setminus S$. This result also shows that the space $\mathcal{X}_{G\setminus(S\cup\{e\})}$ is included in the space $\mathcal{X}_{G\setminus S}$.

Proposition 2 *$\mathcal{X}_{G\setminus(S\cup\{e\})} \subseteq \mathcal{X}_{G\setminus S}$, and furthermore $\mathcal{X}_{G\setminus(S\cup\{e\})} = \mathcal{X}^{\bar{e}}_{G\setminus S} \cup \mathcal{X}^{\emptyset}_{G\setminus S}$.*

Proof Since $S \subset S\cup\{e\}$, the influence graph $G\setminus(S\cup\{e\})$ has one less edge than $G\setminus S$, while other parameters of the two graphs remain the same. This implies that any live-edge graph X generated from the former influence graph can always be generated from the latter one, which establishes the first part of the proposition. Furthermore, $\mathcal{X}_{G\setminus(S\cup\{e\})}$ contains those live-edge graphs without edge e, which is essentially the union of $\mathcal{X}^{\bar{e}}_{G\setminus S}$ and $\mathcal{X}^{\emptyset}_{G\setminus S}$ by definition. ■

4.3 Across Space Mapping

Our third result further divides $\mathcal{X}^{\emptyset}_{G\setminus S}$ into a collection of partitions $\{\Phi_i\}$, and establishes a one-to-one mapping between Φ_i and element X_i in the space $\mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})}$, where $g=(u',v')$ is an edge in $E\setminus S$ with $v'\neq v$.

Proposition 3 *Let $t = |\mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})}|$, then $\mathcal{X}^{\emptyset}_{G\setminus S}$ can be divided into t partitions $\{\Phi_i\}_{i=1}^{t}$ such that, for every Φ_i, these exists a corresponding $X_i \in \mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})}$, and vice versa.*

The live-edge graphs we employ in this proof are illustrated in Figure 4.

Proof We will explicitly construct a set $\Phi_i \subseteq \mathcal{X}^{\emptyset}_{G\setminus S}$ for each element $X_i \in \mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})}$. There are two types of elements in $\mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})}$, and we will construct Φ_i respectively as follows: (1) If node v' has an incoming edge in X_i, then $\Phi_i = \{X_i\}$. Φ_i is contained in $\mathcal{X}^{\emptyset}_{G\setminus S}$ since $\mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})} \subseteq \mathcal{X}^{\emptyset}_{G\setminus S}$ using a similar argument as in the space inclusion

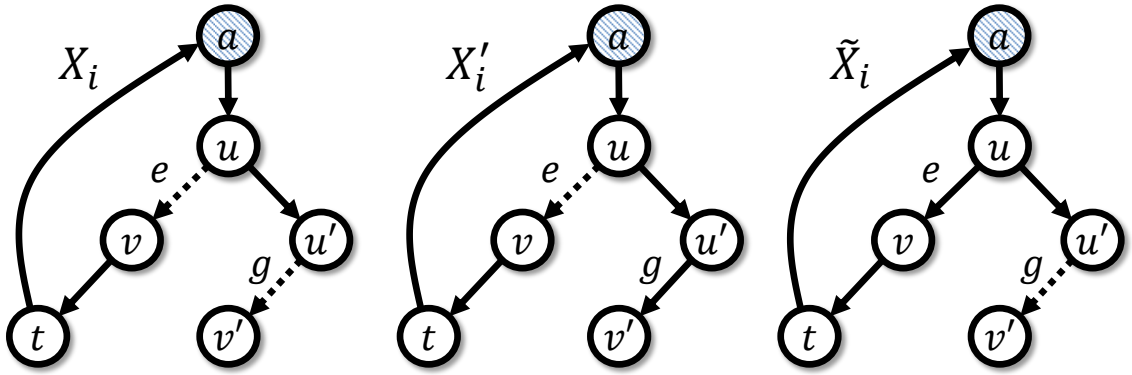


Figure 4: Illustration of the live-edge graphs used in the proof of Proposition 3. A dashed line indicates that the edge has been deleted.

property. (2) Otherwise, $\Phi_i = \{X_i, X_i'\}$, where $X_i' = (V, E_{X_i} \cup \{g\})$ is obtained by extending X_i with edge g. Φ_i is also contained in $\mathcal{X}^{\emptyset}_{G\setminus S}$ since $g \notin S$ and hence X_i' is a valid live-edge graph in $\mathcal{X}^{\emptyset}_{G\setminus S}$. It is easy to see that the Φ_is are pairwise disjoint and form a partition of the space $\mathcal{X}^{\emptyset}_{G\setminus S}$. ■

4.4 Across Space Probability Mapping

Our fourth result relates the probability of the partition $\Phi_i \subseteq \mathcal{X}^{\emptyset}_{G\setminus S}$ to the probability of the corresponding live-edge graph $X_i \in \mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})}$. This result is a sequel to the across space mapping property in the last section. Essentially, we show that the sum of the probabilities of the elements in Φ_i is equal to the probability of X_i.

Proposition 4 *For every $\Phi_i \subseteq \mathcal{X}^{\emptyset}_{G\setminus S}$ and its associated $X_i \in \mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})}$,* $\Pr[X_i|G \setminus (S \cup \{g\})] = \sum_{H\in\Phi_i} \Pr[H|G \setminus S]$.

Proof We will consider two cases. When $\Phi_i = \{X_i\}$ is a singleton, $\Pr[X_i|G \setminus (S \cup \{g\})] = \Pr[X_i|G \setminus S]$ and hence the statement holds true trivially. When $\Phi_i = \{X_i, X_i'\}$, the difference $\Pr[X_i|G \setminus (S \cup \{g\})] - \sum_{H\in\Phi_i} \Pr[H|G \setminus S]$ is proportional to $p(v', X_i, G \setminus (S \cup \{g\})) - p(v', X_i, G \setminus S) - p(v', X_i', G \setminus S)$, where we only need to consider the contribution of the terminal node v' of edge $g = (u', v')$,

since all other nodes contribute the same amount to the probability of each live-edge graph involved. Based on Eq. (2), the difference between the first two terms, $w(u', v')$, cancels out with the third term, $w(u', v')$, which shows that the difference $\Pr[X_i|G \setminus (S \cup \{g\})] - \sum_{H \in \Phi_i} \Pr[H|G \setminus S]$ is zero, and completes the proof. ■

CHAPTER V

NETWORK OPTIMIZATION

In this chapter, we will prove a set of results for the LT model, namely that the objective functions are *supermodular* for all three problems: edge deletion, edge addition and node deletion. These results will form the basis for our algorithm design in Chapter 6.

5.1 Definitions

5.1.1 Supermodularity

A set function $f : 2^E \mapsto \mathbb{R}$ defined over the power set 2^E of a set E is called *supermodular* iff $\forall S \subseteq T \subset E, \forall e \in E \setminus T$

$$f(S \cup \{e\}) - f(S) \leq f(T \cup \{e\}) - f(T). \tag{3}$$

Intuitively, for a monotonic increasing supermodular function f, the marginal gain of adding a new element e to a set T is greater than the gain of adding e to any subset S of T. This property is referred to as the *increasing differences* property, as opposed to *diminishing returns* in the case of a submodular function. If f is a monotonic decreasing function (as will be the case when we consider deleting edges or nodes), then the marginal loss in adding e to T would be smaller than that of adding e to S.

5.1.2 Susceptibility

We define the *susceptibility* of an influence graph G to a set of potential sources A as $\sum_{a \in A} \sigma(a, G)$, which is the sum of the influence function for each node a. Intuitively, one can think of each node $a \in A$ as having equal probability of being the source, and the susceptibility of G is the expected value of the influence function with respect to

the randomness of picking any source a from A. Our definition of susceptibility can also be generalized to the case where each node a has a different probability of being the source. In this case, all our subsequent theorems would still hold. Furthermore, we assume that the size of the source set is only poly-logarithmic in the total number of nodes in the network, *i.e.*, $|A| \ll |V|$.

In our setting, we are interested in manipulating the underlying influence graph in order to minimize or maximize its susceptibility. We will first formulate the *edge deletion* problem and show the monotonicity and supermodularity of its objective function.

5.2 Edge Deletion Problem

In this problem, given an influence graph $G = (V, E, w)$ and a set of sources A, we want to delete a set of edges S^* of size k from G such that the susceptibility of the resulting influence graph is minimized. That is

$$S^* := \operatorname*{argmin}_{S \subseteq E : |S| = k} \sum_{a \in A} \sigma(a, G \setminus S), \tag{4}$$

where the objective function is a set function over the edges S to be deleted. We will show that each $\sigma(a, G \setminus S)$ is a monotonically decreasing and supermodular function of S, and hence their positive sum $\sum_{a \in A} \sigma(a, G)$, also is.

5.2.1 Monotonicity

In this section, we prove that $\sigma(a, G \setminus S)$ is a monotonically decreasing function of S. Since the set of live-edge graphs and the associated probabilities involved in the computation of the influence function will change as edges are deleted, it is not obvious that this function is monotonically decreasing. For instance, if we consider the difference between $\sigma(a, G \setminus S)$ and $\sigma(a, G \setminus (S \cup \{e\}))$, the former function sums over $\mathcal{X}_{G \setminus S}$ whereas the latter sums over $\mathcal{X}_{G \setminus (S \cup \{e\})}$. We will use the within space

mapping property in Proposition 1 and the space inclusion property in Proposition 2 to prove the following result:

Theorem 5 *$\sigma(a, G \setminus S)$ is a monotonically decreasing function of the set of edges S to be deleted.*

Proof Given the influence graph $G = (V, E, w)$, we need to show that for any set $S \subseteq E$ and $e = (u, v) \in E \setminus S$

$$\sigma(a,G \setminus S) - \sigma(a, G \setminus (S \cup \{e\})) \geqslant 0.$$

Using the fact that the space $\mathcal{X}_{G\setminus S}$ is divided into three partitions, $\mathcal{X}^{e}_{G\setminus S}$, $\mathcal{X}^{\bar{e}}_{G\setminus S}$ and $\mathcal{X}^{\emptyset}_{G\setminus S}$, and the space $\mathcal{X}_{G\setminus(S\cup\{e\})}$ is divided into two partitions $\mathcal{X}^{\bar{e}}_{G\setminus S}$ and $\mathcal{X}^{\emptyset}_{G\setminus S}$ (space inclusion property in Proposition 2), we can write the difference as:

$$\begin{aligned}
&\sigma(a, G \setminus S) - \sigma(a, G \setminus (S \cup \{e\})) \qquad (5)\\
&= \sum_{X \in \mathcal{X}^{e}_{G\setminus S}} \Pr[X|G \setminus S] \cdot r(a, X)\\
&+ \sum_{X \in \mathcal{X}^{\emptyset}_{G\setminus S}} (\Pr[X|G \setminus S] - \Pr[X|G \setminus (S \cup \{e\})]) \cdot r(a, X)\\
&+ \sum_{X \in \mathcal{X}^{\bar{e}}_{G\setminus S}} (\Pr[X|G \setminus S] - \Pr[X|G \setminus (S \cup \{e\})]) \cdot r(a, X)
\end{aligned}$$

Recall that $e = (u, v)$. We will simplify the last two summands in the above equation using the following two facts:

- For each $X \in \mathcal{X}^{\emptyset}_{G\setminus S}$ based on Equation (2):

$$\Pr[X|G \setminus S] - \Pr[X|G \setminus (S \cup \{e\})] = -w(u, v) \prod_{v' \neq v} p(v', X, G \setminus S)$$

- For $X \in \mathcal{X}^{\bar{e}}_{G\setminus S}$, the probability is the same:

$$p(v, X, G \setminus S) = p(v, X, G \setminus (S \cup \{e\})) = w(\bar{e})$$

.

Then Equation (5) is equal to:

$$\begin{aligned}
&\sigma(a, G \setminus S) - \sigma(a, G \setminus (S \cup \{e\})) \\
&= \sum_{X \in \mathcal{X}^e_{G \setminus S}} \Pr[X | G \setminus S] \cdot r(a, X) \\
&+ \sum_{X \in \mathcal{X}^\emptyset_{G \setminus S}} -w(u, v) \prod_{v' \neq v} p(v', X, G \setminus S) \cdot r(a, X) + 0
\end{aligned}$$

Since an $\widetilde{X} \in \mathcal{X}^e_{G \setminus S}$ has probability $\Pr[\widetilde{X} | G \setminus S] = w(u, v) \cdot \prod_{v' \neq v} p(v', \widetilde{X}, G \setminus S)$, then using Proposition 1 to match $\widetilde{X} \in \mathcal{X}^e_{G \setminus S}$ to $X \in \mathcal{X}^\emptyset_{G \setminus S}$, we have:

$$\begin{aligned}
&\sigma(a, G \setminus S) - \sigma(a, G \setminus (S \cup \{e\})) \\
&\qquad = \sum_{X \in \mathcal{X}^\emptyset_{E \setminus S}} \Pr[\widetilde{X} | G \setminus S] \cdot \Big(r(a, \widetilde{X}) - r(a, X) \Big) .
\end{aligned} \tag{6}$$

Since the live-edge graph $\widetilde{X}$ has one more edge than X, clearly $r(a, \widetilde{X}) - r(a, X) \geqslant 0$, which completes the proof. ■

5.2.2 Supermodularity

In this section, we will prove that $\sigma(a, G \setminus S)$ is a supermodular function of S. We will use the across space mapping property in Proposition 3 and the probability mapping property in Proposition 1 to prove the following result:

Theorem 6 *The function $\sigma(a, G \setminus S)$ is a supermodular function of the set of edges S to be deleted.*

Proof Given an influence graph $G = (V, E, w)$, $S \subset E$ and $e = (u, v), g = (u', v') \in E \setminus S$, we will establish the supermodularity of $\sigma(a, G \setminus S)$, by showing that:

$$\sigma(a, G \setminus S) - \sigma(a, G \setminus (S \cup \{e\}) \geqslant \sigma(a, G \setminus (S \cup \{g\})) - \sigma(a, G \setminus (S \cup \{g, e\})$$

Let $t = |\mathcal{X}^\emptyset_{G \setminus (S \cup \{g\})}|$, then using the across space mapping property in Proposition 3, we can divide $\mathcal{X}^\emptyset_{G \setminus S}$ into t partitions $\{\Phi_i\}_{i=1}^t$ and rewrite Equation (6) in the proof of

Theorem 5 as:

$$\begin{aligned}&\sigma(a, G\backslash S) - \sigma(a, G \setminus (S \cup \{e\})) \\ &\quad = \sum_{i=1}^{t} \sum_{X \in \Phi_i} \Pr[\widetilde{X}|G \setminus S] \cdot \Big(r(a, \widetilde{X}) - r(a, X)\Big)\end{aligned} \tag{7}$$

Using a similar reasoning to that of Equation (6) in the proof of Theorem 5 for $G \setminus (S \cup \{g\})$, we have:

$$\begin{aligned}&\sigma(a, G\backslash(S \cup \{g\})) - \sigma(a, G \setminus (S \cup \{g, e\}) \\ &\quad = \sum_{X \in \mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})}} \Pr[\widetilde{X}|G \setminus (S \cup g)] \cdot \Big(r(a, \widetilde{X}) - r(a, X)\Big)\end{aligned} \tag{8}$$

Then, we need only compare Equation (8) and (7) term by term for each $X_i \in \mathcal{X}^{\emptyset}_{G\setminus(S\cup\{g\})}, i = 1, \ldots, t$. Clearly, when $\Phi_i = \{X_i\}$, the terms from the two equations are equal. When $\Phi_i = \{X_i, X_i'\}$, we need to show that:

$$\begin{aligned}&\Pr[\widetilde{X}_i|G \setminus S] \cdot \Big(r(a, \widetilde{X}_i) - r(a, X_i)\Big) \\ &\qquad + \Pr[\widetilde{X}_i'|G \setminus S] \cdot \Big(r(a, \widetilde{X}_i') - r(a, X_i')\Big) \\ &\qquad \geqslant \Pr[\widetilde{X}_i|G \setminus (S \cup \{g\})] \cdot \Big(r(a, \widetilde{X}_i) - r(a, X_i)\Big)\end{aligned} \tag{9}$$

Based on the probability mapping property in Proposition 4, we have $\Pr[\widetilde{X}_i|G \setminus (S \cup \{g\})] = \Pr[\widetilde{X}_i|G \setminus S] + \Pr[\widetilde{X}_i'|G \setminus S]$. Then to establish Equation (9), it suffices to show that

$$r(a, \widetilde{X}_i') - r(a, X_i') \geqslant r(a, \widetilde{X}_i) - r(a, X_i).$$

Recall that $X_i' = (V, E_{X_i} \cup \{g\})$. Since live-edge graphs are constructed in a way that each node has at most one incoming edge, each reachable node y has a unique path from the source node a to node y. Furthermore, (1) a reachability path in $\widetilde{X}_i$ is clearly also present in $\widetilde{X}_i'$. Therefore if removing edge $e = (u, v)$ from $\widetilde{X}_i$ results in unreachability of some nodes in X_i then those same nodes become unreachable when removing e from $\widetilde{X}_i'$; (2) removing edge e from $\widetilde{X}_i'$ may disconnect some additional

nodes whose paths from the source a include edge g. Hence, the reduction in reachable nodes when removing edge e from $\widetilde{X}'_i$ is the same or larger than the reduction when removing edge e from $\widetilde{X}_i$. This completes the proof. ■

This result under the LT model is quite surprising. In fact, under the independent cascade model, Sheldon et al. [39] even provided a counter-example showing that the objective function is not submodular. We show that, under the independent cascade model, the objective is not supermodular either, by a simple counter-example illustrated in Fig. 5.

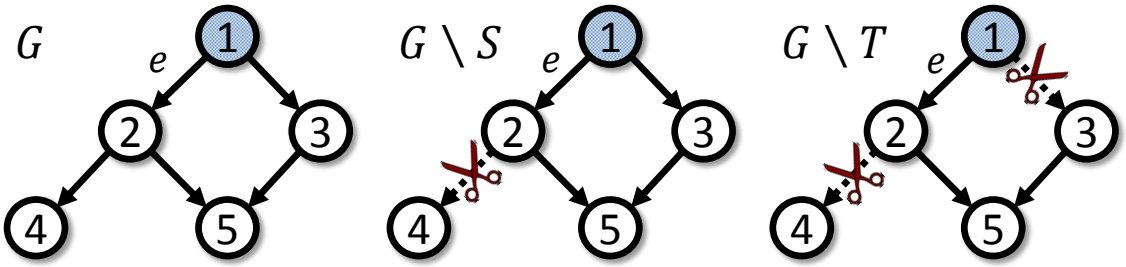


Figure 5: Counter-example where the IC model is not supermodular. Node 1 is the source, and all edge probabilities are all ones. The marginal loss resulting from adding e to S is 1, whereas that resulting from adding e to T is 2, which violates the supermodularity inequality.

Recall that under the *Independent Cascade* (IC) model, starting with a set of activated nodes A at time $t = 0$, at each discrete time step $t = 1, 2, \ldots$ each newly activated node v is given a single attempt at activating each of its still inactive neighbors u with probability of success w_{vu}, independently of previous activation attempts. If v succeeds, then u is newly activated at time $t + 1$.

Theorem 7 *The function $\sigma(a, G \setminus S)$ is not a supermodular of S under the Independent Cascade model.*

Proof We give a counter-example to prove the above. Consider the graph illustrated in Fig. 5(a) as our original influence graph $G = (V, E, w)$ with all weights equal to 1. Hence, in this trivial setting there is always only one possible cascade.

Let $S = \{(2,4)\}, T = S \cup \{(1,3)\}$, and $e = (1,2)$. The resulting graphs after removing S and T are illustrated in Fig. 5. The influence of node 1 after removing S is 3 (1 can reach nodes 2, 3, 5), and adding e to S results in an influence of 2, and so the marginal loss of adding e to S is 1. The influence of node a after removing T is 2, and adding e to T reduces the influence to 0, with a marginal loss of 2. Hence adding e to the smaller set S results in a smaller marginal loss, violating the supermodularity property. ■

5.3 Edge Addition Problem

In this section, given a *partial* influence graph $G'(V, E', w)$ and a larger *potential* influence graph $G = (V, E, w)$ with $E' \subseteq E$, we want to add to G' a set of edges S^* of size k from $E \setminus E'$ such that the resulting susceptibility is maximized:

$$S^* := \operatorname*{argmax}_{S' \subseteq E \setminus E' : |S'| = k} \sum_{a \in A} \sigma(a, G' \cup S'), \tag{10}$$

where $G' \cup S' := (V, E' \cup S', w)$, and the objective function is a set function over the edges S' to be added. We will show that each $\sigma(a, G' \cup S')$ is monotonic and supermodular, and hence their positive combination is also monotonic and supermodular.

Theorem 8 *The function $\sigma(a, G' \cup S')$ is a monotonic and supermodular function of the set of S' edges to be added.*

Proof We will prove the results by relating the objective function to that of the edge deletion problem and then apply the results from the edge deletion problem. More specifically, let $S' \subseteq T'$ and $e \in E \setminus T'$. If we define $S = E \setminus (S' \cup E' \cup \{e\})$, then

$$\sigma(a, G' \cup (S' \cup \{e\})) = \sigma(a, G \setminus S)$$
$$\sigma(a, G' \cup S') = \sigma(a, G \setminus (S \cup \{e\})),$$

since $G' \cup (S' \cup \{e\}) = G \setminus S$ and $G' \cup S' = G \setminus (S \cup \{e\})$. Similarly, if we define $T = E \setminus (T' \cup E' \cup \{e\})$, then

$$\sigma(a, G' \cup (T' \cup \{e\})) = \sigma(a, G \setminus T)$$
$$\sigma(a, G' \cup T') = \sigma(a, G \setminus (T \cup \{e\})).$$

Note that $S' \subseteq T'$ implies that $T \subseteq S$. Then we apply the supermodularity of $\sigma(a, G \setminus S)$ as a function of the edges S to be deleted in Theorem 6, and obtain

$$\sigma(a, G' \cup (S' \cup \{e\})) - \sigma(a, G' \cup S')$$
$$\leqslant \sigma(a, G' \cup (T' \cup \{e\})) - \sigma(a, G' \cup T'),$$

which completes the proof. ■

5.4 Node Deletion Problem

In this problem, given an influence graph $G = (V, E, w)$ and a set of sources A, we want to delete a set of nodes U^* of size k and all edges incident to these nodes from G such that the susceptibility of the resulting influence graph is minimized. That is

$$U^* := \operatorname*{argmin}_{U \subseteq V : |U| = k} \sum_{a \in A} \sigma(a, G \setminus E_U), \tag{11}$$

where E_U is the set of all edges incident to nodes U. Here the objective function is a set function over the nodes U to be deleted. We note that we do not need to explicitly remove the nodes U from the influence graph since $p(u, X, G \setminus E_U)$ for these nodes are always 1, not affecting the influence function. We will show that the objective function is monotonic and supermodular by reducing it to the edge deletion problem.

Theorem 9 *The function $\sigma(a, G \setminus E_U)$ is a monotonic and supermodular function of the set of nodes U to be deleted.*

Proof Deleting a set of nodes $U \subseteq V$ is equivalent to deleting the set of all edges E_U incident to these nodes. For another set of nodes $U' \subseteq V$ where $U \subseteq U'$, it must hold $E_U \subseteq E_{U'}$. Monotonicity is then obvious.

Now for an additional node $u \in V$ and $u \notin U'$, we will consider the effect of deleting the the edges $E_{\{u\}}$ incident to u. Thus, we will divide $E_{\{u\}}$ into three disjoint partitions, S_1, S_2 and S_3, and show that for $i = 1, 2, 3$

$$\begin{aligned} &\sigma(a, G \setminus (E_U \cup S_i)) - \sigma(a, G \setminus E_U) \\ &\qquad \leqslant \sigma(a, G \setminus (E_{U'} \cup S_i)) - \sigma(a, G \setminus E_{U'}). \end{aligned} \tag{12}$$

More specifically,

- $S_1 := E_{\{u\}} \cap E_U$. Relation (12) holds since both its sides are 0.
- $S_2 := (E_{\{u\}} \cap E_{U'}) \setminus E_U$. Since $E_{U'} \cup S_2 = E_{U'}$, the right-hand side of relation (12) is 0. But its l.h.s. is negative using the monotonicity of the edge deletion problem. Hence the relation also holds.
- $S_3 := E_{\{u\}} \setminus E_{U'}$. Relation (12) holds using the supermodularity of the edge deletion problem.

Last, we can combine the three parts by incrementally deleting S_1, S_2 and S_3, leading to

$$\begin{aligned} &\sigma(a, G \setminus (E_U \cup E_{\{u\}})) - \sigma(a, G \setminus E_U) \\ &\qquad \leqslant \sigma(a, G \setminus (E_{U'} \cup E_{\{u\}})) - \sigma(a, G \setminus E_{U'}), \end{aligned}$$

which proves the theorem. ■

Adding nodes is also supermodular by a similar proof based on g_a instead of f_a. However, we do not treat this case as it is less amenable to real scenarios.

CHAPTER VI

ALGORITHMS

Given that the objective functions for the edge (node) deletion (addition) problems are supermodular, we can, in principle, solve the network topology optimization problems using the state-of-the-art supermodular optimization algorithms. However, there remain great challenges in scaling these algorithms up to diffusion networks with millions of nodes. First, supermodular optimization requires evaluating the influence function many times. The problem of computing the influence function $\sigma(a, G)$ exactly has been shown to be #P-Hard [9]. Thus there is a need to design methods to approximately compute the influence function in sublinear time. To tackle this problem, we will estimate $\sigma(a, G)$ using *empirical averaging* (EA) over a *fixed* set of live-edge graphs, pre-sampled using the LT live-edge graph generation process described in Chapter 3.2. That is

$$\sigma(a, G) \approx \widehat{\sigma}(a, G) := \frac{1}{|\mathcal{L}|} \cdot \sum_{X_i \in \mathcal{L}} r(a, T^a_{X_i}) \tag{13}$$

where $\mathcal{L} = \{X_i\}_{1 \leq i \leq l}$ is the set of sampled *live-edge graphs* from G, and $T^a_{X_i}$ is the tree rooted at a induced from X_i.

Second, typically, the marginal change of the influence function for each candidate edge or node needs to be computed. This imposes the additional requirement that each marginal change computation has to be nearly constant-time to handle the large number of candidate edges. We will address these challenges as follows

- **Edge** & **Node deletion.** We will design an efficient descendant-counting tree data structure which can be constructed in linear time and supports constant time queries on the influence function. We will use this data structure as part of a greedy algorithm to minimize the supermodular objective function.

- **Edge addition.** We will employ an efficient randomized neighbor-counting graph data structure which can also be constructed in linear time and supports constant time queries on the influence function. We will use this data structure inside a modular approximation algorithm to maximize the supermodular objective function.

6.1 Edge Deletion

It is easy to see that the empirical average influence function $\widehat{\sigma}(a, G \setminus S)$ under edge deletion is also supermodular and monotonically decreasing. The classical result by Nemhauser et. al.[33] shows that minimizing a supermodular function [1] with cardinality constraints can be approximated to a constant factor of $(1-1/e)$ using a simple greedy approach: at each iteration, given the current solution S_t, add to the solution the element e with the largest *marginal loss* $\Delta(e|S_t)$ defined using Equation (13)

$$\frac{1}{\mathcal{L}} \sum_{a \in A} \sum_{X_i \in \mathcal{L}} r(a, T^a_{X_i} \setminus S_t) - r(a, T^a_{X_i} \setminus (S_t \cup \{e\})) \tag{14}$$

where $T^a_{X_i} \setminus S_t$ means deleting edges S_t from tree $T^a_{X_i}$. Based on this expansion, we notice that the edge with largest marginal loss is the edge whose deletion results in the largest decrease in the average number of descendants over all source nodes and the set of induced live-edge trees T^a_X. Note that we use the terminology "marginal loss" rather than "marginal gain" because our objective function is monotone *decreasing*, hence $\Delta(e|S_t)$ measures the marginal loss resulting from removing e after edges S_t have been removed.

Naïvely applying the greedy algorithm is computationally intensive and will not scale to networks with millions of nodes. Basically, at iteration t, for every edge $e \in E \setminus S_t$ and every T^a_X, we need to compute $\Delta(e|S_t)$ by performing a Breadth-First-Search (BFS) traversal from the source a, and count the number of node reachable

[1] [33] deals with the problem of maximizing a submodular function, which is equivalent to minimizing a supermodular function.

in $T^a_{X_i} \setminus S_t$ and $T^a_{X_i} \setminus (S_t \cup \{e\})$. It is easy to see that such an approach will lead to an $O(|V|^2 + |V||E|)$ complexity algorithm: BFS is $O(|V| + |E|)$ and we need to check $O(|E|)$ edges in $E \setminus S_t$. Such quadratic dependence on the network size motivates us to design a more efficient solution.

6.1.1 Scaling Up

Can we avoid the many BFS traversals? To answer this question, we first make the following observation

Observation 1 *Given an edge $e = (u, v)$ to be deleted where v is reachable from the source a, the marginal loss $\Delta(e|S_t)$ can be computed as*

$$r(a, T^a_X \setminus S_t) - r(a, T^a_X \setminus (S_t \cup \{e\})) = r(v, T^a_X \setminus S_t) + 1 \tag{15}$$

This observation implies that, if we can compute $r(v, T^a_X)$ for all $v \in V$ in the original live-edge tree T^a_X, we can then compute the marginal gain of each edge e efficiently.

Can we compute the number of descendants, $r(v, T^a_X)$, efficiently, for *all* $v \in V$? Fortunately, since we are dealing with trees of at most $|V|$ edges each, this can be done in time $O(|V|)$ using a single BFS traversal. More specifically, after initializing $r(v, T^a_X) = 0,\ \forall v \in V$,

1. Perform a BFS starting from the source a of T^a_X, adding each traversed edge e to a stack H; at the end of the BFS, the top of the queue is the last edge traversed.
2. While stack H is not empty, pop edge $e = (u, v)$ and increment $r(u, T^a_X)$ by $r(v, T^a_X) + 1$.

The correctness of the above procedure is easy to verify: the number of descendants of a node is equal to the sum of the number of descendants of its children, plus the number of children it has, which is exactly what we are computing.

Suppose we have already maintained the descendant counts $r(v, T_X^a \setminus S_t)$ for all node $v \in V$. Then after deleting edge $e = (u, v)$, there are two types of nodes for which need to update the descendant counts: the ancestors u' of node v, and the nodes that have become unreachable. For the former, we update their descendant counts by subtracting out the number of descendants of node v plus 1. Similar to Equation (15),

$$r(u', T_X^a \setminus (S_t \cup \{e\})) = r(u', T_X^a \setminus S_t) - r(v, T_X^a \setminus S_t) - 1.$$

For the latter, we simply set their descendant counts to zero, *i.e.*, $r(u', T_X^a \setminus (S_t \cup \{e\})) = 0$. Last, the marginal loss of each edge can also be updated according to Equation (15).

We illustrate the descendant-counting tree data structure in Figure 6.

6.1.2 Overall Algorithm: GreedyCuttingEdges

The overall algorithm GreedyCuttingEdges is summarized in Algorithm 1. It first samples live-edge graphs and obtains the corresponding live-edge trees for the input sources A. Line 4–12 compute the initial descendant counts variables $r(u, T_X^a)$ for each node u and each T_X^a, and the edge marginal loss variables $\Delta(e)$ for all edges in E. For each iteration, line 15 adds to the solution set the edge with largest marginal loss, and lines 16–27 locally update the descendant count variables for nodes, and marginal loss variables for edges. Finally, the solution set S^* is returned.

6.1.3 Time & Space complexity

If we assume the number of source nodes to be poly-logarithmic in $|V|$, then algorithm 1 has computational complexity $O(k|\mathcal{L}||V|)$ which is linear (up to poly-logarithimic factors) in the size of the network. As for space complexity, our main data structures store the node descendant counts for each induced live-edge tree on

one hand, and the marginal losses of the edges on the other requiring a space complexity of $O(|E| + |V||\mathcal{L}|)$, linear in the network size.

6.2 Node Deletion

The node deletion algorithm follows exactly from the edge deletion algorithm we have just described. Simply put, instead of measuring the *marginal loss* at the level of edges, we measure it at the level of nodes, and follow the greedy strategy of iteratively adding the node with largest marginal loss to the solution set. This slight modification of the algorithm does not have any complexity or implementation consequences, besides modifying the bookkeeping of descendant counts from edges to nodes. These modifications are summarized in Algorithm 2, which will be referred to as GreedyCuttingNodes.

6.3 Edge Addition

We now turn to our algorithmic framework for solving the problem of adding edges. Recently, Iyer et al.[21] proposed a simple approach for constrained submodular minimization with approximation guarantees, which we will adapt for our (analog) supermodular maximization problem. The algorithm constructs a *modular lower bound* (MLB) of the objective function, and then adds edges that maximize this lower bound, instead of the original objective. That is,

$$\frac{1}{|\mathcal{L}|}\sum_{a\in A}\sum_{X_i\in\mathcal{L}} r(a, T^a_{X_i}\cup S) \geqslant \frac{1}{|\mathcal{L}|}\sum_{a\in A}\sum_{X_i\in\mathcal{L}}\sum_{e\in S} r(a, T^a_{X_i}\cup\{e\}),$$

where $T^a_{X_i}\cup S$ refers to the live-edge tree resulting from adding new edges S to $T^a_{X_i}$. Note that the resulting tree may allow the source a to reach some nodes originally not reachable in $T^a_{X_i}$.

The MLB approach has several nice properties:

1. The modular lower bound function is simple, essentially requiring us to compute the reachability score $r(a, T^a_{X_i}\cup\{e\})$ for each candidate edge to be added.

ALGORITHM 1: GreedyCuttingEdges

Input: Influence Graph $G(V, E, w)$, Sources A, k
Output: Edges S^*

Sample a set of live-edge graphs $\mathcal{L} = \{X\}$ from G
Obtain the set of induced live-edge trees $\{T_X^a\}$ from $\mathcal{L}$
Initialize $\Delta(e) = 0$ for all $e \in E$, $r(u, T_X^a) = 0$ for all $u \in V$ and T_X^a
for *each* T_X^a **do**
    Initialize queue Q, stack H, $Q.enqueue(a)$, $visited = \{a\}$
    while Q *is not empty* **do**
        $s = Q.dequeue()$
        for $u \in V$ *and* (s, u) *is an edge in* T_X^a **do**
            if $u \notin visited$ **then**
                $visited = visited \cup \{u\}$, $Q.enqueue(u)$, $H.push((s, u))$
    while H *is not empty* **do**
        $(u, v) = H.pop()$, $r(u, T_X^a)$ += $r(v, T_X^a) + 1$, $\Delta((u, v))$ += $r(v, T_X^a) + 1$
$S^* = \emptyset$
for *t=1 to* k **do**
    $e_t = (u_t, v_t) = \text{argmax}_{e \in E \setminus S^*} \Delta(e)$, $S^* = S^* \cup \{e_t\}$
    for *each* T_X^a **do**
        $s = u_t$
        while s *is not the source* a **do**
            $r(s, T_X^a)$ −= $r(v_t, T_X^a) + 1$, $\Delta((parent(s), s))$ −= $r(v_t, T_X^a) + 1$
            $s = parent(s)$
        Initialize queue Q, $Q.enqueue(u_t)$, $visited = \{u_t\}$
        while Q *is not empty* **do**
            $s = Q.dequeue()$
            for $u \in V$ *and* (s, u) *is an edge in* T_X^a **do**
                if $u \notin visited$ **then**
                    $visited = visited \cup \{u\}$, $Q.enqueue(u)$
                    $\Delta((s, u))$ −= $r(u, T_X^a) + 1$, $r(s, T_X^a) = 0, r(u, T_X^a) = 0$
return S^*

2. Maximizing the MLB for a budget k reduces to simply finding the top k edges which lead to the largest function value

$$\sum_{a \in A} \widehat{\sigma}(a, G \cup \{e\}) = \frac{1}{|\mathcal{L}|} \sum_{a \in A} \sum_{X_i \in \mathcal{L}} r(a, T_{X_i}^a \cup \{e\}). \tag{16}$$

3. The MLB algorithm is guaranteed to obtain an approximation solution within a factor of $1/(1 - \kappa_\sigma))$ of the optimum where κ is the curvature of our objective function (see Thm.5.9 in [22]).

ALGORITHM 2: GreedyCuttingNodes

Input: Influence Graph $G(V, E, w)$, Sources A, k
Output: Nodes S^*

Sample a set of live-edge graphs $\mathcal{L} = \{X\}$ from G
Obtain the set of induced live-edge trees $\{T_X^a\}$ from $\mathcal{L}$
Initialize $\Delta(e) = 0$ for all $e \in E$, $r(u, T_X^a) = 0$ for all $u \in V$ and T_X^a
for *each* T_X^a **do**
    Initialize queue Q, stack H, $Q.enqueue(a)$, $visited = \{a\}$
    while Q *is not empty* **do**
        $s = Q.dequeue()$
        for $u \in V$ *and* (s, u) *is an edge in* T_X^a **do**
            if $u \notin visited$ **then**
                $visited = visited \cup \{u\}$, $Q.enqueue(u)$, $H.push((s, u))$
    while H *is not empty* **do**
        $(u, v) = H.pop()$, $r(u, T_X^a)$ += $r(v, T_X^a) + 1$, $\Delta(u)$ += $r(v, T_X^a) + 1$
$S^* = \emptyset$
for *t=1 to k* **do**
    $n_t = \text{argmax}_{n \in V \setminus S^*} \Delta(n)$, $S^* = S^* \cup \{n_t\}$
    for *each* T_X^a **do**
        $s = n_t$
        while s *is not the source* a **do**
            $r(s, T_X^a)$ −= $r(v_t, T_X^a) + 1$, $\Delta(s)$ −= $r(v_t, T_X^a) + 1$
            $s = parent(s)$
        Initialize queue Q, $Q.enqueue(u_t)$, $visited = \{n_t\}$
        while Q *is not empty* **do**
            $s = Q.dequeue()$
            for $u \in V$ *and* (s, u) *is an edge in* T_X^a **do**
                if $u \notin visited$ **then**
                    $visited = visited \cup \{u\}$, $Q.enqueue(u)$
                    $\Delta(u)$ −= $r(u, T_X^a) + 1$, $r(s, T_X^a) = 0, r(u, T_X^a) = 0$
return S^*

However, naïvely applying the MLB algorithm is computationally intensive and can not be scaled up to networks with millions of nodes. Basically, for every candidate edge e to be added and for every T_X^a, we need to compute the reachability $r(a, T_{X_i}^a \cup \{e\})$ by performing a Breadth-First-Search (BFS) traversal from the source a, and count the number of node reachable in $T_{X_i}^a \cup \{e\}$. It is easy to see that such an approach will lead to an $O(|V||E|)$ complexity algorithm: BFS is $O(|V|)$ (since

trees T_X^a have at most $|V|$ edges), and we need to check $O(|E|)$ edges in $E \setminus E'$. Such quadratic dependence on the network size motivates us to design a more efficient solution.

6.3.1 Scaling Up

Can we avoid the many BFS traversals? To answer this question, we first make the following observation

Observation 2 *Given an edge $e = (u, v)$ to be added, where node v is originally not reachable from the source a, but becomes reachable with the addition of e, the reachability of a can be updated as*

$$r(a, T_X^a \cup \{e\}) = r(a, T_X^a) + w(e) \cdot (r(v, \overline{T}_X^a) + 1) \tag{17}$$

where $\overline{T}_X^a$ is the complement of T_X^a containing those nodes and edges not reachable from a.

We note that the term $r(v, \overline{T}_X^a) + 1$ is multiplied by the weight $w(e)$ of edge e to account for the *probability* of that edge being actually picked by node v in the live-edge generation process of the new influence graph $G \cup \{e\}$. Furthermore, note that $\overline{T}_X^a$ may contain cycles.

Can we compute the number of reachable nodes, $r(v, \overline{T}_X^a)$, efficiently, for all v in $\overline{T}_X^a$? Fortunately, this problem has been extensively studied in the theoretical computer science literature as the neighborhood size-estimation problem[11], and was recently applied in the context of influence estimation for a continuous-time diffusion model[13]. We will adapt a linear time algorithm by Cohen[11] for our problem.

We apply the algorithm to $\overline{T}_X^a$ as follows: first, we assign to each node u a label $l(u)$ drawn from the exponential distribution with parameter (mean) 1. Then, we exploit the fact that the minimum $l^*(v)$ of the set of exponential random labels $\{l(u)\}$

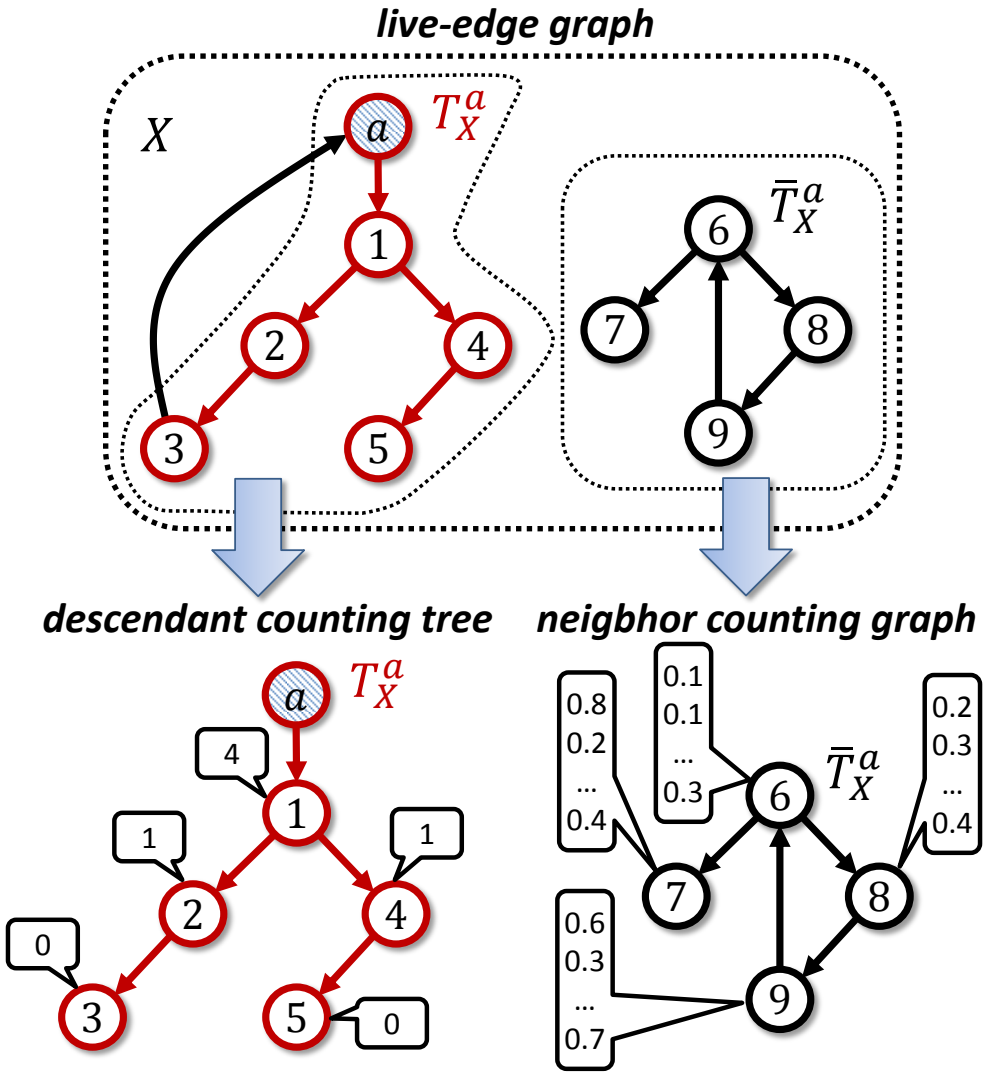


Figure 6: Illustration of a live-edge graph X, the tree T_X^a induced by BFS rooted at a, the part of the graph $\overline{T}_X^a$ which is the complement to T_X^a. For the edge deletion problem, we build a descendant counting tree data structure for each live-edge tree T_X^a where each node stores its number of descendants. For the edge addition problem, we build a neighbor-counting graph data structure for each complement $\overline{T}_X^a$, where each node stores a list of q least labels $\{l_i^*(v)\}$ obtained over q random labelings.

for nodes u reachable from v will itself be an exponential random variable, with its parameter equal to $r(v, \overline{T}_X^a)$. If we repeat the random labeling q times and obtain q such *least labels* $\{l_i^*(v)\}_{i=1}^q$, then the neighborhood size is estimated as

$$r(v, \overline{T}_X^a) \approx \frac{q-1}{\sum_{i=1}^q l_i^*(v)}. \tag{18}$$

Can we find the least labels efficiently for all nodes v given each random labeling? In fact, this can be done using a modified BFS traversal which requires time only linear in the network size. More specifically, for a given labeling, we start from the node v with the smallest label, and perform a BFS traversal in the reverse direction of the graph edges. For each node u encountered in the BFS, we set $l^*(u) = l(v)$. Once a node has been encountered in a BFS and its least label has been set, it is marked as *visited*, not only for this particular BFS, but across all subsequent BFS runs. After the BFS traversal from node v is complete, we move to the *unvisited* node with the next smallest label, and repeat the procedure iteratively until all nodes have been marked as visited.

It is easy to see why this algorithm correctly assigns the appropriate least label $l^*(v)$ to each node v: since we order the BFS runs by minimum labels, and are traversing the edges in reversed direction, then once a node u has been visited, we are guaranteed that the $l^*(u)$ we assign to it is the smallest, and any subsequent BFS that can reach u will have a larger label than $l^*(u)$.

6.3.2 Overall Algorithm: ModularAdding

We now summarize the algorithm ModularAdding in 3: we first generate the live-edge graphs, induce the live-edge trees, and draw q labels for each node $v \in V$ from the exponential distribution with mean 1 (lines 1-5). Then, for each source node $a \in A$, we iterate over the induced live-edge trees T_X^a, collecting the estimated neighborhood size of each node v in the complement $\overline{T}_X^a$ of each such tree, by applying the Cohen algorithm (lines 7-20). After having iterated over the live-edge trees, we compute the

final score for each edge $e \in \mathcal{C}$ in the candidate set $\mathcal{C}$ as the sum over v's neighborhood size estimates, weighted by the edge's diffusion probability $w(e)$ (lines 21-22). Finally we sort the scores vector in descending order, and return the top k edges.

6.3.3 Time & Space complexity

We assume the number of source $|A|$ is poly-logarithmic in the number of node $|V|$ and hence ignored in the complexity analysis. Then algorithm 3 has computational complexity $O(q|\mathcal{L}||V|)$ and space complexity $O(q|V|)$. This is because Cohen's algorithm has complexity $O(q|V|)$. This algorithm is invoked $O(|\mathcal{L}|)$ times. The final sorting of the scores can be done in $O(|E|)$. As for space, we only require data structures of sizes linear in the number of nodes $O(q|V|)$ to hold the least labels.

ALGORITHM 3: ModularAdding

Input: $G(V, E, w), k$, Sources A, Candidates $\mathcal{C}$

Output: Edges S^*

Randomly generate a set of live-edge graphs $\mathcal{L}$ from G
Obtain the set of induced live-edge trees $\{T_X^a\}$ from $\mathcal{L}$
for *each* $v \in V$ **do**
    for *each* $i = 1, \ldots, q$ **do**
        $l_i(v) \sim \exp(-x)$
for *each* $a \in A$ **do**
    for *each* $\overline{T}_X^a$ **do**
        for *each* $i = 1, \ldots, q$ **do**
            $visited = \emptyset$
            for *nodes* v *in* $\overline{T}_X^a$ *ordered according to* $argsort(\{l_i(v)\})$ **do**
                if $v \notin visited$ **then**
                    $visited = visited \cup \{v\}$
                    Initialize Q, $Q.enqueue(v)$
                    while Q *is not empty* **do**
                        $u = Q.dequeue()$, $l_i^*(u) = l(v)$
                        $visited = visited \cup \{u\}$
                        for *each parent* s *of* u *in* $\overline{T}_X^a$ **do**
                            $Q.enqueue(s)$
        for *each node* v *in* $\overline{T}_X^a$ **do**
            $r(v, \overline{T}_X^a) = \dfrac{q-1}{\sum_{i=1}^{q} l_i^*(v)}$
    for *each* $e = (u, v) \in \mathcal{C}$ **do**
        $\texttt{score}(e)$ += $w(e) \cdot \sum_{X_i \in \mathcal{L}} (r(v, \overline{T}_X^a) + 1)$
$S^* = argsort(\texttt{score}, k, \text{descending})$
return S^*

CHAPTER VII

EXPERIMENTS

In this section, we describe our experimental setting and present results for the edge deletion, node deletion and edge addition problems. We will introduce the datasets we use, briefly explain the competing heuristics, and present the results along the following axes:

- *Efficacy*: how well do our algorithms fare in terms of minimizing and maximizing influence, as compared to the heuristics?
- *Scalability*: how do our algorithms scale as the sizes of the networks grow?

7.1 Experimental setup

7.1.1 Synthetic networks

We generate three types of networks using the Kronecker graph model [30], which is known to generalize a number of realistic graph models: The parameters used to generate the networks are presented in Table 1.

1. CorePeriphery: the name of this model is due to its structure, whereby some nodes form a densely connected core, while other nodes fall at the sparsely connected periphery [3].
2. ErdosRenyi: a classical random graph model, where an edge is established between every pair of nodes with a certain fixed probability, independently of all other pairs of nodes [14]. Although this model may not be suitable for representing real networks, it remains crucial for the analysis of graphs and their structure.

Table 1: The parameter matrix used as input the Kronecker network model to generate the synthetic networks.

Dataset	Parameter Matrix
CorePeriphery	[0.9 0.5; 0.5 0.3]
ErdosRenyi	[0.5 0.5; 0.5 0.5]
Hierarchical	[0.9 0.1; 0.1 0.9]

Table 2: Datasets summary. Top section contains synthetic networks. Numbers outside the bracket are for edge deletion experiments; those inside the bracket are for edge addition experiments. Bottom section contains real-world networks statistics.

Dataset	#Nodes	#Edges
CorePeriphery ErdosRenyi Hierarchical	1M(65K)	2M (131K)
HepPH	35K	420K
Epinions	75K	509K
MemeTracker	0.8K	5K

3. Hierarchical: a more modern model that attempts to produce random graphs that are *scale-free* and exhibit a high degree of clustering [35]. This model has been shown to mimic the structure of real networks such as the World-wide Web and the Internet (at the domain level).

These three graph models have very different structural properties, allowing us to test for sensitivity to network structure. Additionally, we generate synthetic networks (specifically CorePeriphery) of various sizes in order to test for the scalability of our algorithms.

7.1.2 Real-world networks

We choose three publicly available real-world datasets [1] that are amenable to diffusion processes and hence suitable for our problems:

[1] http://snap.stanford.edu/data/

Table 3: Parameter values used in the experiments of Figs. 7 and 8 (first row in table), and Figures 11 and 12 (second row): A is the set of sources, $\mathcal{L}_{\texttt{opt}}$ is the set of live-edge graphs used by our algorithm, $\mathcal{L}_{\texttt{eval}}$ is the set of live-edge graphs used for evaluation of all algorithms and heuristics, t refers to the budget of edges deleted for which diffusion stops completely, q is the number of random labelings used in algorithm 3.

	Parameters for Experiments				
Problem	$\lvert A\rvert$	$\lvert\mathcal{L}_{\texttt{opt}}\rvert$	$\lvert\mathcal{L}_{\texttt{eval}}\rvert$	k	q
Edge Deletion	100	1,000	5,000	$[0, t]$	–
Edge Addition				$[0, 2000]$	20

1. HepPH: a whom-cites-whom citation network based on the Arxiv *High-energy Physics* papers over the period 1993-2003; an edge (u, v) exists if paper v cites paper u

2. Epinions: a who-trusts-whom online social network of the consumer review site Epinions.com; an edge (u, v) exists if user v declares that he trusts user u

3. MemeTracker: a who-copies-from-whom network of news media sites and blogs, where each node u is a news media site or blog, and edge (u, v) represents the recorded event of v copying u. These edges are inferred from actual hyperlink cascade traces using a network inference algorithm, NetInf [18]

The statistics of the datasets are summarized in Table 2.

7.1.3 Assigning probabilities

Given a network $G(V, E)$, we populate the weight vector representing the probabilities on the edges E according to the LT model, as follows:

1. for a given node $v \in V$, we draw a probability value $\widetilde{w}(u, v)$ for each edge $e = (u, v) \in E$ that is incoming into v, uniformly at random from the interval $[0, 1]$.

2. In addition, we draw from the same interval a probability value $\overline{w}_v$ representing no infection, i.e the probability that v's infected parents fail to activate it.

3. Since the probabilities on the edges plus the probability of no infection must sum to 1, we then normalize each probability over the sum of all the probabilities, *i.e.*, we obtain $w(u,v) = \overline{w}(u,v)/(\sum_{u \in V} w(u,v) + \overline{w}_v)$, and a similar normalized value for $\overline{w}_v = \dfrac{\overline{w}_v}{\sum_{u \in V} w(u,v) + \overline{w}_v}$.

We apply this method for all datasets except for MEMETRACKER.

For the MEMETRACKER dataset, we make use of the median transmission time, also provided as part of the dataset. Let $\widetilde{t}(u,v)$ be the median transmission time between two nodes u and v, then we set $w(u,v) \propto \widetilde{t}(u,v)^{-1}$, rewarding smaller transmission times with higher diffusion probabilities, and vice versa. We assign a probability of $\overline{w}_v = 0.2, \forall v \in V$, and normalize the weights for all nodes v such that $\sum_{u \in V} w(u,v) + \overline{w}_v = 1$.

7.1.4 Competing heuristics

As we have emphasized earlier, the network optimization problems we address under the LT model are, to our knowledge, novel, and hence algorithms that solve these same problems have not been proposed in the literature. Consequently, in order to evaluate the efficacy of the solutions provided by our algorithms, we compare against other heuristic measures that are not based on the dynamics entailed by the LT diffusion model. Briefly, the heuristics we propose have either been used for similar problems under different diffusion models, or in a rather unprincipled way.

7.1.4.1 Description

These heuristic strategies can be described as follows:

1. `Random`: select k edges (nodes) uniformly at random from the input set of edges (nodes). This is a simple baseline measure. To evaluate the objective functions at a given set of edges (nodes) (of size k) that are chosen at random, we generate 5 such sets, and present the average value across these sets.

2. `Weights`: select the k edges with highest diffusion probability (weight) $w(u, v)$. This measure is applicable to the edge addition/deletion problems only. It allows us to measure the impact of our algorithms exploiting the cascade trees, as opposed to simply myopically choosing edges with highest diffusion probability, both for edge deletion and addition.

3. `Betweenness`: select k edges (nodes) with highest *edge (node) betweenness centrality* [4]. Intuitively, this measure captures the centrality of an edge or node relative to the shortest paths between pairs of nodes. Betweenness centrality has been suggested as a measure for immunizing nodes against diseases spreading under SIR-type models [38].

4. `Eigen`: select the k edges (nodes) that cause the maximum decrease (increase) in the leading eigenvalue of the network when removed from it, or added to it [43, 41]. This spectral measure has been shown to be provably effective under the SIR diffusion model, as it exploits the relationship between the *leading eigenvalue* of the network, and its *epidemic threshold*, i.e. the configuration of SIR parameters at which an outbreak is inevitable [43].

5. `Degree`: select the k edges whose destination nodes have the highest out-degrees, or the k nodes have the highest out-degrees. For edges, this measure effectively deletes (adds) edges that link to *hubs*, or nodes with high degree centrality. For node deletion, it favors nodes with high degree centrality. This heuristic has been used in an experimental study on node immunization in email networks in [16], also under an SIR-type model of diffusion.

7.1.4.2 Further notes on competing heuristics

For edge deletion, we initially allowed the heuristics to choose the top k edges to delete among *all existing edges*. However in practice, we observed an extremely weak

performance across all heuristics, due to the fact that they delete edges without any consideration for the source set A. To correct for that, we restrict all heuristics to deleting edges whose source nodes are in A, i.e. $\{e = (a, v) \in E : a \in A\}$. This restriction makes the efficacy of the heuristics' solutions at a similar scale as ours.

For node deletion, and similarly to edge deletion, we also restrict the set of nodes eligible for deletion to the source nodes, as the results were much poorer in the unrestricted case (where all nodes in V can be chosen).

For edge addition, the heuristics are given the set of candidate edges as input, and so they choose the top k edges according to the metric in question.

Note that for heuristics `Eigen` and `Degree`, the network is weighted, where the diffusion probability of each edge is also its weight in the corresponding adjacency matrix. For `Betweenness`, the network is also weighted with the *inverse* $1/w(u, v)$ of the diffusion probability $w(u, v)$ of an edge (u, v); a higher probability $w(u, v)$ is transformed into a smaller edge weight, resulting in this edge participating in more shortest-paths and hence a higher `Betweenness` score.

We also point out that we for the larger datasets we experiment on, we estimate `Betweenness` by running the single-source shortest-paths computations from the set of source nodes used for that experiment as in [5], rather than all n nodes, given its $O(|V^2| \log |V| + |V||E|)$ complexity.

7.2 Deleting Edges

We carry out each experiment as follows: given an influence graph $G(V, E, w)$, a set of source nodes A chosen uniformly at random from V, and a budget k of edges to delete, we run GreedyCuttingEdges and the five heuristics and obtain a set of edges from each. Then, for each algorithm or heuristic, we simulate the LT diffusion process by generating a set of live-edge graphs $\mathcal{L}_{\texttt{eval}}$ based on G, and then deleting the proposed set of edges S^* from all live-edge graphs in $\mathcal{L}_{\texttt{eval}}$. The efficacy of each

proposed set of edges is measured by I_k, the average number of infected nodes over $\mathcal{L}_{\texttt{eval}}$. These parameters are summarized in Table 3. The budget k is increased until diffusion is no longer possible, *i.e.*, the source nodes are completely isolated.

7.2.1 Synthetic networks

The results are shown in 7. First, we observe that our algorithm clearly outperforms *all five* other heuristics: for *any* budget k of edges to delete, our algorithm minimizes the graph susceptibility ratio (I_k/I_0) better than any of the heuristics for all three synthetic network types, implying that it produces good solutions *independently* of the structural properties of the input network. On the other hand, the considered heuristics perform arbitrarily good or bad, as we vary the type of synthetic network. At last, we observe that even for $|\mathcal{L}_{opt}| = 1,000$, a quantity much smaller than the typical $10,000$ used in the literature, the green and red lines are almost indistinguishable, meaning our solution generalizes well to the larger evaluation set of $5,000$ live-edge graphs.

7.2.2 Real-world networks

We observe similar results for real-world networks. For instance, for the EPINIONS dataset (Fig. 8(b)), our method has decreased the graph susceptibility to 40% of its original value at $k = 200$, whereas the best performing heuristic at the same k is `Weights` with 60% (the lower the better here).

7.3 Deleting Nodes

We carry out our node deletion experiments in a way that is similar to those for edge deletion as described in 7.3.

7.3.1 Synthetic networks

The results for node deletion on synthetic networks are illustrated in Figure 9. Again, our algorithm outperforms all four other heuristics consistently for all three types

of synthetic networks. However, one can notice that heuristics `Degree` and `Eigen` perform only slightly worse than GreedyCuttingNodes. This can be explained by the fact that the search space for these heuristics, which is restricted to source nodes of A, is considerably smaller than that of the edge deletion experiments (the set of all edges outgoing from the source nodes), resulting in a much narrower gap. However, we are still able to obtain better reduction in graph susceptibility, consistently across all three synthetic networks.

7.3.2 Real-world networks

As in the synthetic setting, our algorithm for deleting nodes is the most successful at mitigating diffusion, for all three real-world networks that we consider. It is particularly interesting to note the behavior of some of the competing heuristics, such `Eigen` and `Betweenness`: while `Eigen` outperforms `Betweenness` for the MemeTracker dataset (Figure 10 c), the former heuristic's performance degrades at around $k = 50$ for the Epinions dataset, relative to the latter heuristic. Even more striking is the arbitrarily bad performance of `Eigen` and `Betweenness` for the HepPH dataset (Figure 10 a). Quite contrarily, our algorithm's performance is unchanged and insensitive to the various datasets, validating our strong mathematical basis.

7.4 Adding Edges

The experimental procedure for evaluating our ModularAdding algorithm and other heuristics is analogous to that described in 7.2 for edge deletion. Recall that the heuristics are given a set of candidate edges as input, and will choose the top k edges according to their respective metric. We compare our algorithm to all previously described heuristics, for the exception of the `Betweenness` heuristic, as it is not obvious how meaningful it would be to compute this metric for edges that do not initially exist in the network.

7.4.1 Synthetic networks

Our algorithm is almost always *twice as effective* as the next best heuristic, be it `Weights` or `Degree`. This efficacy gap is consistent across all three types of networks, confirming yet again the robustness of the solutions we find to varying structural properties of networks.

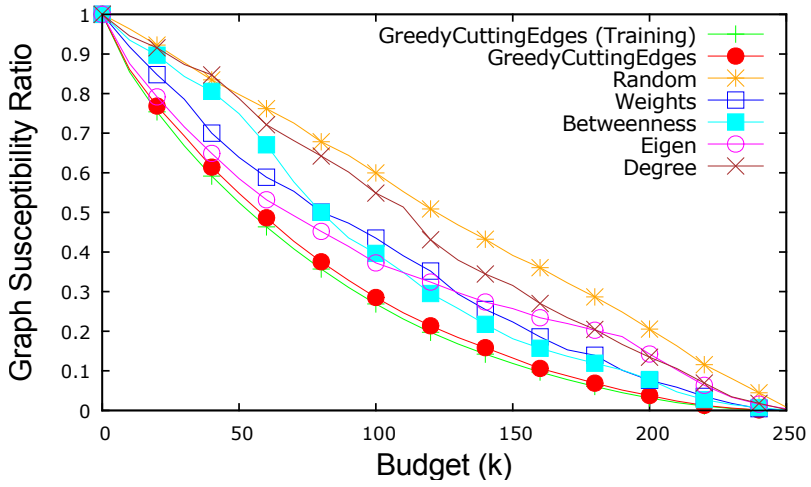
7.4.2 Real-world networks

Similarly to the synthetic setting, our algorithm significantly outperforms all four heuristics in the real-world setting, for all three datasets. For instance, for the HepPH dataset in Fig. 11(a), the `Degree` heuristic requires adding $2,000$ edges to the set A of 100 sources in order to increase the graph susceptibility by twice its initial value (*i.e.*, at $k = 0$), whereas our algorithm increases the graph susceptibility by the same amount for $k = 200$ edges, a small fraction of 2000. This superior performance we obtain implies that our algorithm for adding edges is more amenable to real-world applications, where the budget is typically very small relative to the number of nodes, possibly representing humans in a social network, blogs on the web, etc.

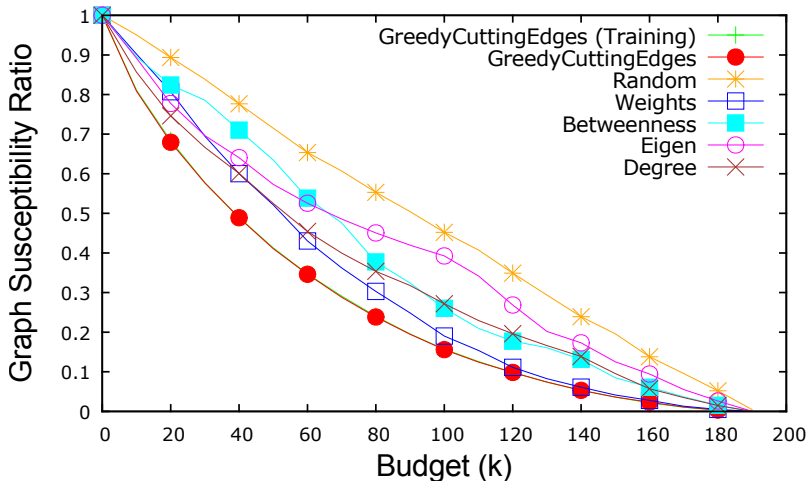
7.5 Scalability

Scalability is a major concern in the industrial setting. We experimentally verify the scalability of both our edge deletion and addition algorithms. All experiments were executed on a Windows 7 laptop with a 2.7GHz quad-core i7 CPU and 16Gb RAM. The results presented in Fig. 13 measure the runtime of our algorithms on synthetic CorePeriphery networks of increasing number of nodes, and fixed average degree of 2. We vary the number of nodes, starting at $2^7 = 128$ nodes, and up to $2^{23} = 8,388,608$ nodes (and $16,777,216$ edges). The experimental results show that our algorithms scale linearly to the size of the network. As expected, the edge addition algorithm, while also having a linear scaling, is more time-consuming than

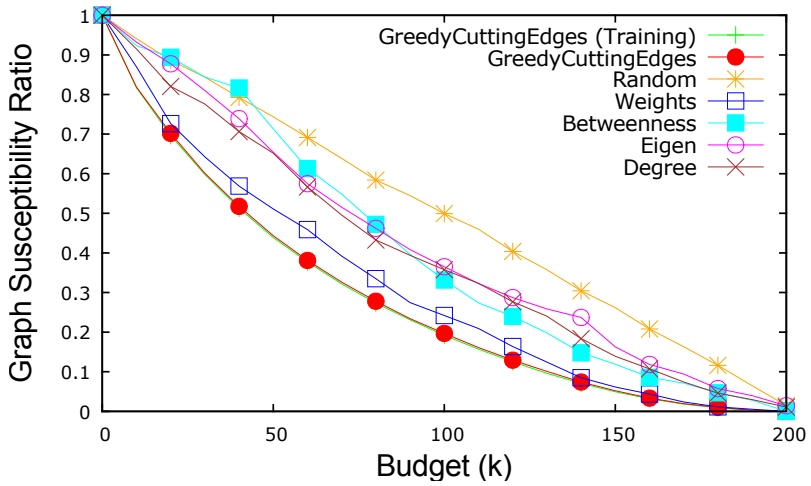
the edge deletion algorithm, due to the repeated linear-time algorithm for building the neighbor-counting graph.



a) CorePeriphery

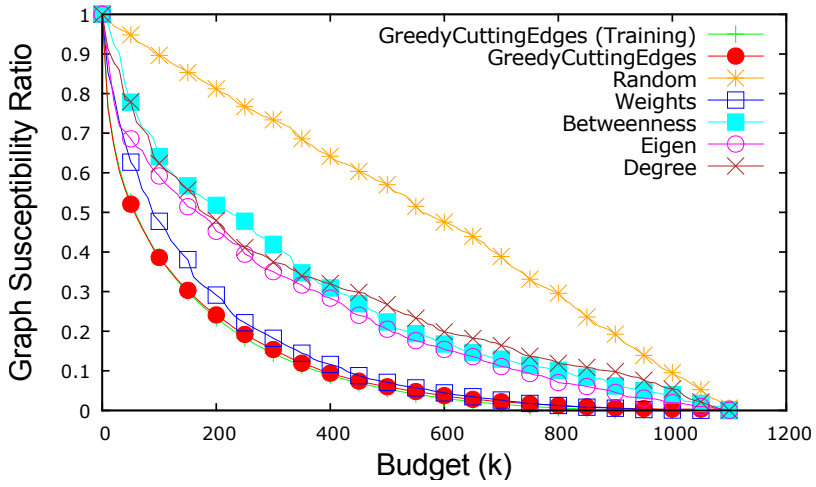


b) ErdosRenyi

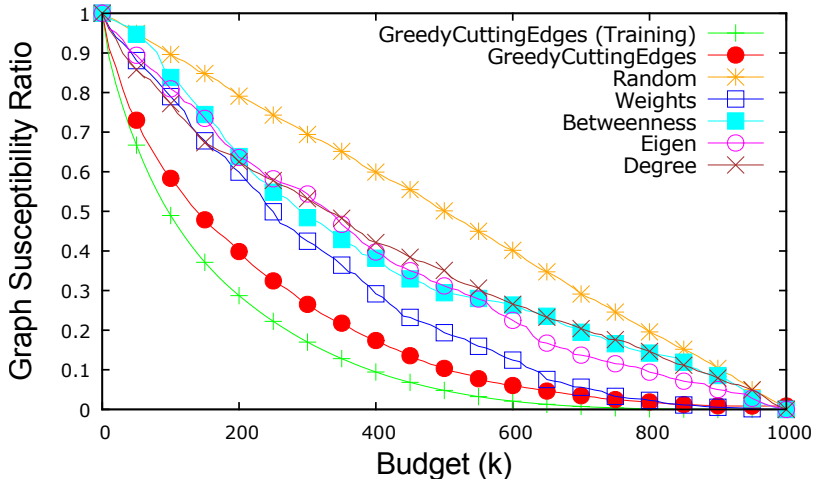


c) Hierarchical

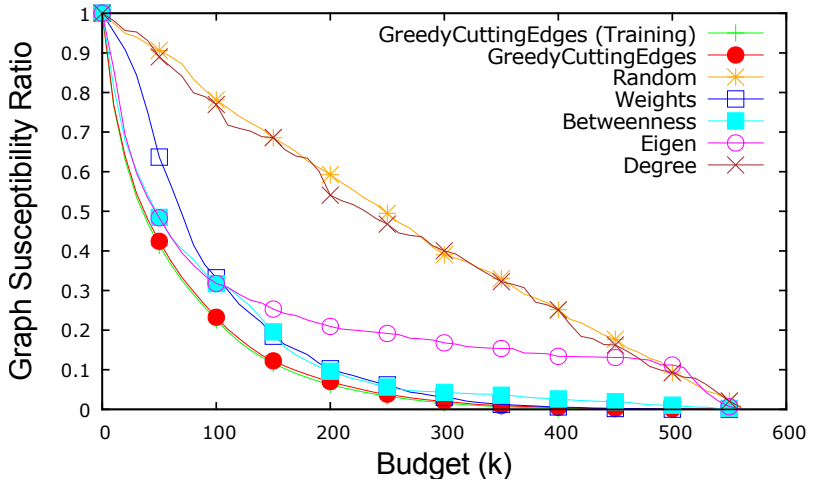
Figure 7: Efficacy of the edge deletion solutions provided by different algorithms for synthetic datasets. Lower is better. The x-axis refers to the budget k; the y-axis refers to the *graph susceptibility ratio*, defined in the range $[0, 1]$ as: I_k/I_0.



a) HepPH

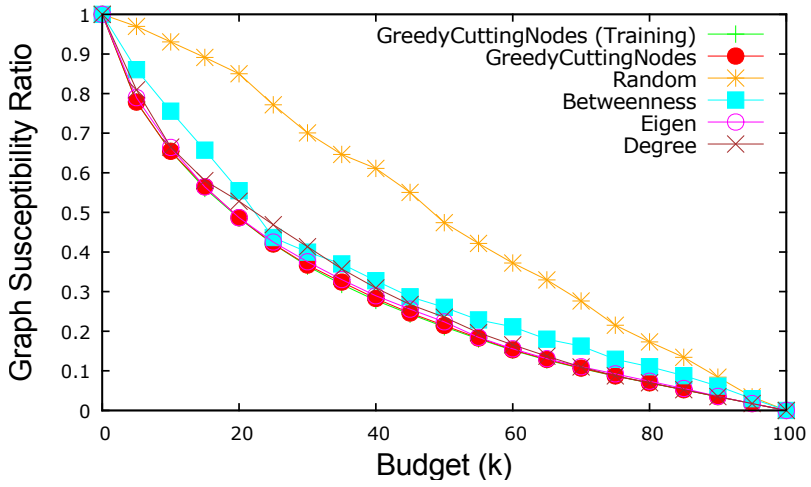


b) Epinions

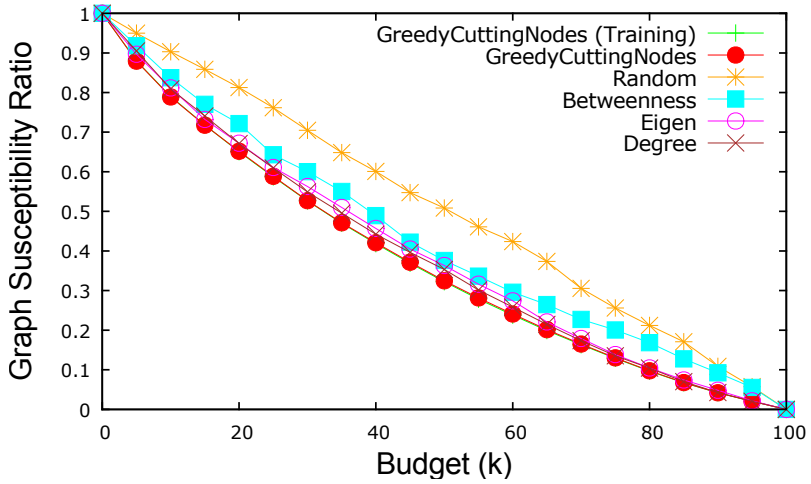


c) MemeTracker

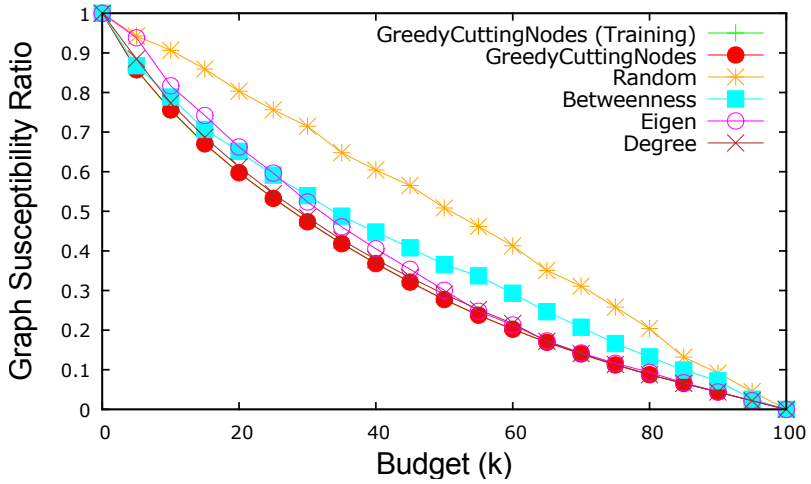
Figure 8: Efficacy of the edge deletion solutions provided by different algorithms for real-world datasets. Lower is better. The x-axis refers to the budget k; the y-axis refers to the *graph susceptibility ratio*, defined in the range $[0, 1]$ as: I_k/I_0.



a) CorePeriphery

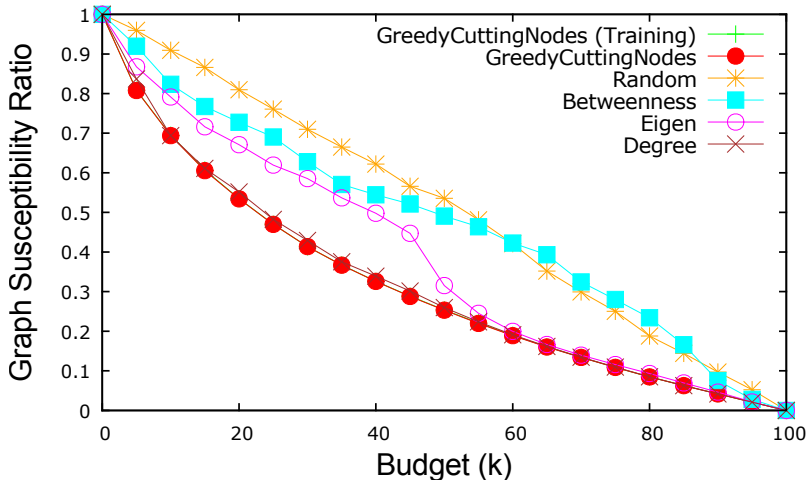


b) ErdosRenyi

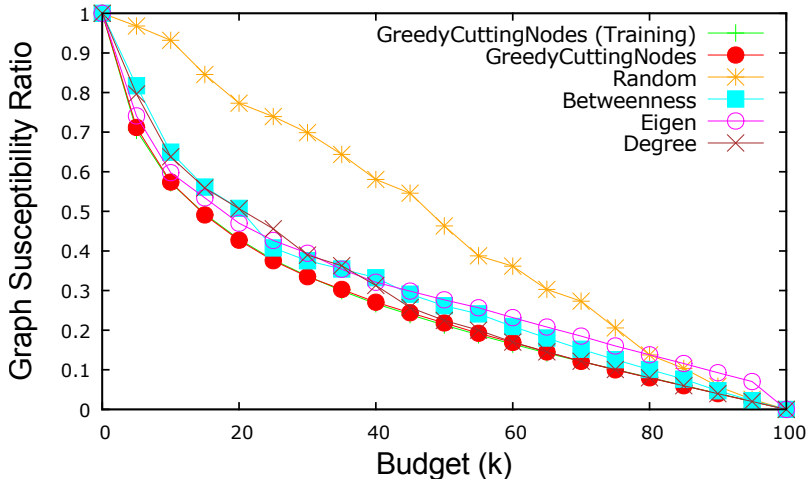


c) Hierarchical

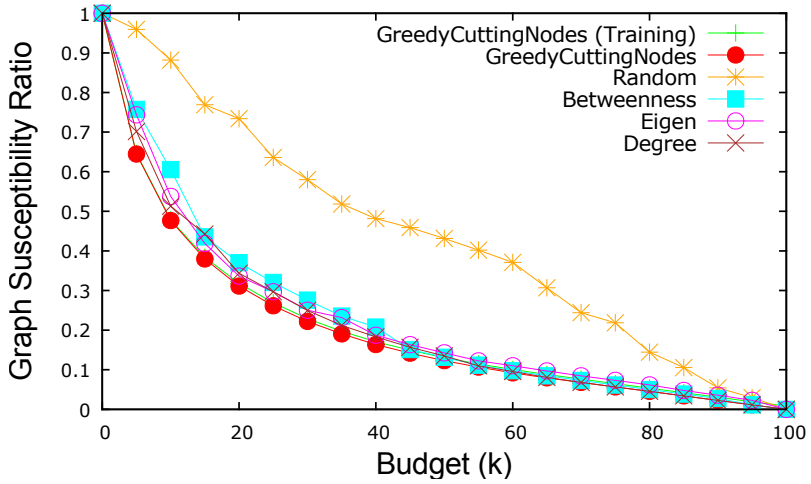
Figure 9: Efficacy of the node deletion solutions provided by different algorithms for synthetic datasets. Lower is better. The x-axis refers to the budget k; the y-axis refers to the *graph susceptibility ratio*, defined in the range $[0, 1]$ as: I_k/I_0.



a) HepPH

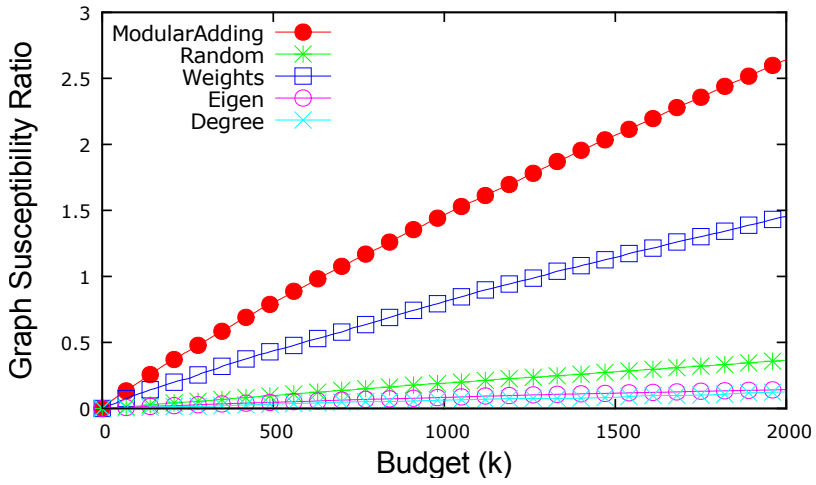


b) Epinions

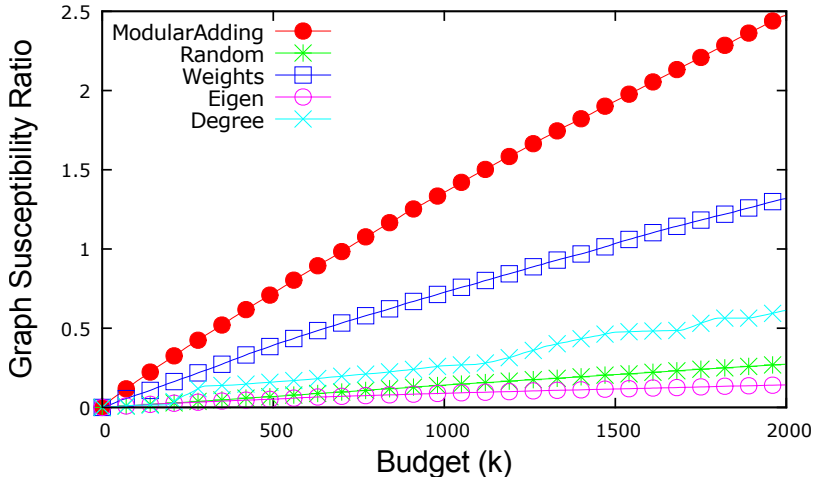


c) MemeTracker

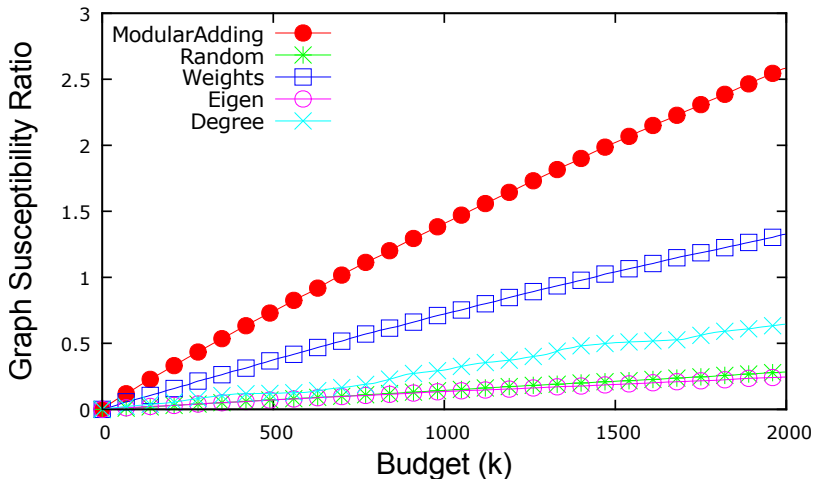
Figure 10: Efficacy of the node deletion solutions provided by different algorithms for real-world datasets. Lower is better. The x-axis refers to the budget k; the y-axis refers to the *graph susceptibility ratio*, defined in the range $[0, 1]$ as: I_k/I_0.



a) CorePeriphery

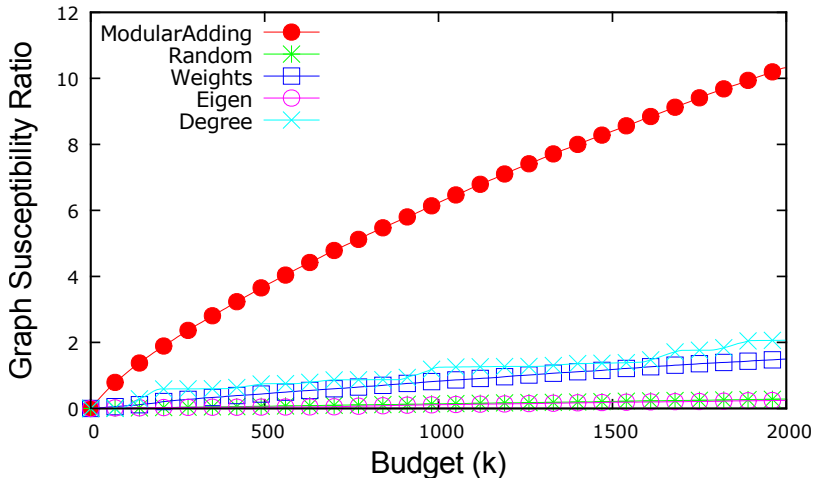


b) ErdosRenyi

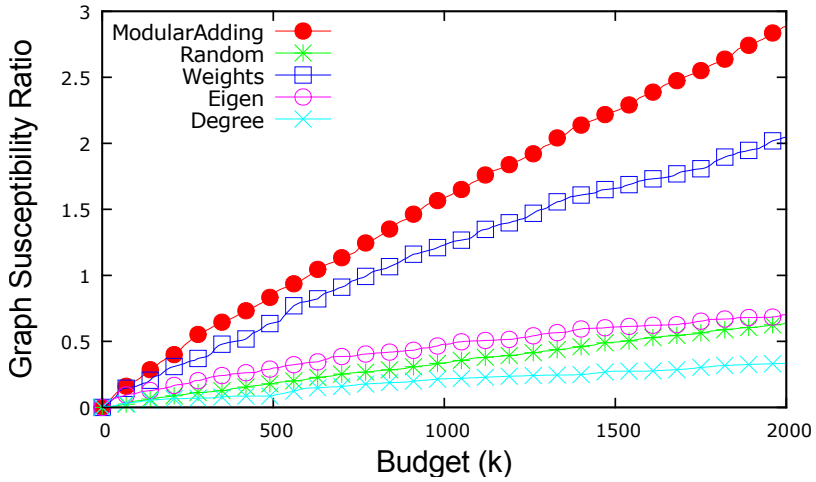


c) Hierarchical

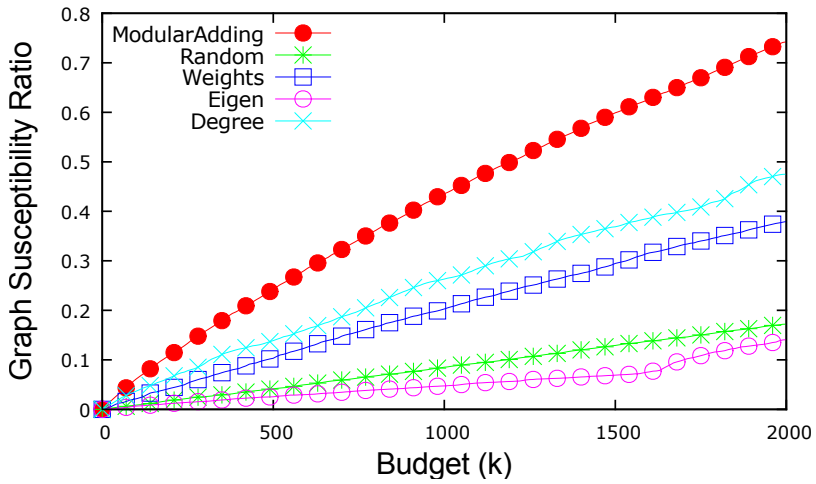
Figure 11: Efficacy of the edge addition solutions provided by different algorithms for synthetic datasets. Higher is better. The x-axis refers to the budget of edges to add; the y-axis refers to the *graph susceptibility ratio*, defined in the range $[0, n]$ as: $(I_k - I_0)/I_0$.



a) HepPH



b) Epinions



c) MemeTracker

Figure 12: Efficacy of the edge addition solutions provided by different algorithms for real-world datasets. Higher is better. The x-axis refers to the budget of edges to add; the y-axis refers to the *graph susceptibility ratio*, defined in the range $[0, n]$ as: $(I_k - I_0)/I_0$.

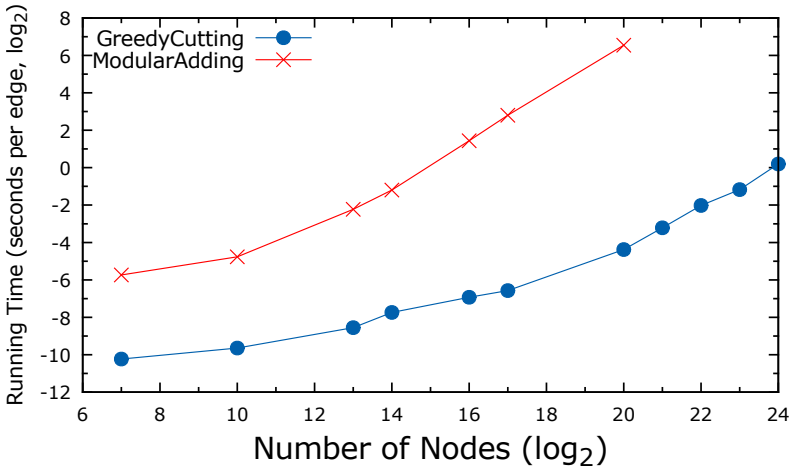


Figure 13: Runtime on synthetic core-periphery networks: for both lines, each point represents the average time in seconds per edge or node deleted (blue), or edge added (red). The number of sources is polylogarithmic in the number nodes and the number of live-edge graphs used is 100, for both problems. For ModularAdding, $q = 5$.

CHAPTER VIII

CONCLUSION

8.1 Contributions

We have investigated the problem of diffusion-aware optimization of network topology under the linear threshold model.

At the theoretical level, we analyzed the mathematical properties of the LT model, and were able to prove, for the first time, the supermodularity of the objective functions for edge addition, edge deletion and node deletion.

Algorithmically, we presented linear time (up to poly-log factors) and space algorithms for edge deletion and addition. Our algorithms employ efficient data structures and randomized subroutines, without which quadratic time solutions would be inevitable. Additionally, the optimization frameworks which we have adapted come with approximation guarantees.

At the experimental level, we demonstrated the efficacy and scalability of both our algorithms. While we significantly outperform heuristics that are based solely on network structure for both problems, we are also able to handle networks with millions of nodes and edges.

8.2 Impact

To our knowledge, this work is the first to address the problem of optimizing networks of diffusion under the widely adopted Linear Threshold model. Hence, we hope that our work would spark interest in this crucial topic in the data mining community, and would trigger additional research into diffusion-aware network optimization. In terms of applying our solutions to real problems, we are confident that under reasonably

simple modeling assumptions, our algorithms would be able to achieve good and non-trivial results. In terms of optimization, we note that our work is (to our knowledge) the first to make use of supermodular functions in the context of networks, serving as a "practitioner's guide" for solving supermodular optimization problems in this area.

8.3 Future Research Directions

Many research directions seem interesting at this point:

- **Learning the LT parameters**: As in most studies, our experimental procedure included randomly generating the edge weights (probabilities). However, we strongly believe that these parameters could be learnt from diffusion cascades that are observed in reality, yielding algorithm outputs that are more meaningful and interpretable.

- **Topology optimization under other models**: While the results regarding optimization under the IC model are mostly negative as we've shown, it is still worth a task to examine other possible solutions for this model, given its wide applicability. It is also worth considering different information propagation frameworks such as voting-based opinion models, which are also of current interest to the community.

- **Competitive diffusion processes**: How do we approach these optimization tasks under multiple competing diffusion processes? While there has been some focus on this topic recently, the formulations and proposed solutions remain limited and impractical. However, social network platforms are often faced with advertising requests from competing brands, politicians, etc., which leaves this type of problems open to research.

REFERENCES

[1] Bakshy, E., Hofman, J. M., Mason, W. A., and Watts, D. J., "Everyone's an influencer: quantifying influence on twitter," in *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 65–74, ACM, 2011.

[2] Bogunovic, I., "Robust protection of networks against cascading phenomena," Master's thesis, ETHZ, 2012.

[3] Borgatti, S. P. and Everett, M. G., "Models of core/periphery structures," *Social networks*, vol. 21, no. 4, pp. 375–395, 2000.

[4] Brandes, U., "A faster algorithm for betweenness centrality*," *Journal of Mathematical Sociology*, vol. 25, no. 2, 2001.

[5] Brandes, U. and Pich, C., "Centrality estimation in large networks," *IJBC*, 2007.

[6] Cai, L., "Fixed-parameter tractability of graph modification problems for hereditary properties," *Information Processing Letters*, vol. 58, no. 4, pp. 171–176, 1996.

[7] Chen, W., Collins, A., Cummings, R., Ke, T., Liu, Z., Rincon, D., Sun, X., Wang, Y., Wei, W., and Yuan, Y., "Influence maximization in social networks when negative opinions may emerge and propagate.," in *SDM*, pp. 379–390, 2011.

[8] Chen, W., Lu, W., and Zhang, N., "Time-critical influence maximization in social networks with time-delayed diffusion process.," in *AAAI*, 2012.

[9] Chen, W., Wang, C., and Wang, Y., "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *ACM KDD*, 2010.

[10] Chen, W., Yuan, Y., and Zhang, L., "Scalable influence maximization in social networks under the linear threshold model," in *IEEE ICDM*, pp. 88–97, 2010.

[11] Cohen, E., "Size-estimation framework with applications to transitive closure and reachability," *Journal of Computer and System Sciences*, vol. 55, no. 3, pp. 441–453, 1997.

[12] Domingos, P. and Richardson, M., "Mining the network value of customers," in *ACM KDD*, pp. 57–66, 2001.

[13] Du, N., Song, L., Zha, H., and Gomez Rodriguez, M., "Scalable influence estimation in continuous time diffusion networks," in *NIPS*, 2013.

[14] Erdős, P. and Rényi, A., "On the evolution of random graphs," *Magyar Tud. Akad. Mat. Kutató Int. Közl*, vol. 5, pp. 17–61, 1960.

[15] Feige, U., Mirrokni, V. S., and Vondrak, J., "Maximizing non-monotone submodular functions," *SIAM Journal on Computing*, vol. 40, no. 4, pp. 1133–1153, 2011.

[16] Gao, C., Liu, J., and Zhong, N., "Network immunization and virus propagation in email networks: experimental evaluation and analysis," *Knowledge and information systems*, vol. 27, no. 2, pp. 253–279, 2011.

[17] Ghosh, A. and Boyd, S., "Growing well-connected graphs," in *Decision and Control, 2006 45th IEEE Conference on*, pp. 6605–6611, IEEE, 2006.

[18] Gomez Rodriguez, M., Leskovec, J., and Krause, A., "Inferring networks of diffusion and influence," in *ACM KDD*, pp. 1019–1028, 2010.

[19] Hethcote, H. W., "The mathematics of infectious diseases," *SIAM review*, vol. 42, no. 4, pp. 599–653, 2000.

[20] Iwata, S., "Submodular function minimization," *Mathematical Programming*, vol. 112, no. 1, pp. 45–64, 2008.

[21] Iyer, R., Jegelka, S., and Bilmes, J., "Fast semidifferential-based submodular function optimization," in *ICML*, 2013.

[22] Iyer, R. K., Jegelka, S., and Bilmes, J. A., "Curvature and optimal algorithms for learning and minimizing submodular functions," in *NIPS*, 2013.

[23] Kempe, D., Kleinberg, J., and Tardos, É., "Maximizing the spread of influence through a social network," in *ACM KDD*, pp. 137–146, ACM, 2003.

[24] Kimura, M., Saito, K., and Motoda, H., "Solving the contamination minimization problem on networks for the linear threshold model," in *PRICAI*, 2008.

[25] Kimura, M., Saito, K., and Motoda, H., "Blocking links to minimize contamination spread in a social network," *ACM TKDD*, vol. 3, no. 2, p. 9, 2009.

[26] Kuhlman, C. J., Tuli, G., Swarup, S., Marathe, M. V., and Ravi, S., "Blocking simple and complex contagion by edge removal," in *IEEE ICDM*, 2013.

[27] Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., and Glance, N., "Cost-effective outbreak detection in networks," in *Conference on Knowledge Discovery and Data Mining* (Berkhin, P., Caruana, R., and Wu, X., eds.), pp. 420–429, ACM, 2007.

[28] Leskovec, J., Adamic, L. A., and Huberman, B. A., "The dynamics of viral marketing," *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, p. 5, 2007.

[29] Leskovec, J., Backstrom, L., and Kleinberg, J., "Meme-tracking and the dynamics of the news cycle," in *ACM KDD*, pp. 497–506, ACM, 2009.

[30] Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z., "Kronecker graphs: An approach to modeling networks," vol. 11, no. Feb, pp. 985–1042, 2010.

[31] Leskovec, J., McGlohon, M., Faloutsos, C., Glance, N. S., and Hurst, M., "Patterns of cascading behavior in large blog graphs.," in *SDM*, vol. 7, pp. 551–556, SIAM, 2007.

[32] Lewis, J. M. and Yannakakis, M., "The node-deletion problem for hereditary properties is np-complete," *Journal of Computer and System Sciences*, vol. 20, no. 2, pp. 219–230, 1980.

[33] Nemhauser, G., Wolsey, L., and Fisher, M., "An analysis of the approximations for maximizing submodular set functions," *Mathematical Programming*, vol. 14, 1978.

[34] Peng, S., Yu, S., and Yang, A., "Smartphone malware and its propagation modeling: A survey," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–17, 2013.

[35] Ravasz, E. and Barabási, A.-L., "Hierarchical organization in complex networks," *Physical Review E*, vol. 67, no. 2, p. 026112, 2003.

[36] Rogers, E. M., *Diffusion of innovations*. Simon and Schuster, 2010.

[37] Rose, D. J. and Tarjan, R. E., "Algorithmic aspects of vertex elimination on directed graphs," *SIAM Journal on Applied Mathematics*, vol. 34, no. 1, pp. 176–197, 1978.

[38] Schneider, C. M., Mihaljev, T., Havlin, S., and Herrmann, H. J., "Suppressing epidemics with a limited amount of immunization units," *Physical Review E*, vol. 84, no. 6, 2011.

[39] Sheldon, D., Dilkina, B., Elmachtoub, A. N., Finseth, R., Sabharwal, A., Conrad, J., Gomes, C. P., Shmoys, D., Allen, W., Amundsen, O., and others, "Maximizing the spread of cascades using network design," *UAI*, 2010.

[40] Sviridenko, M., "A note on maximizing a submodular set function subject to a knapsack constraint," *Operations Research Letters*, vol. 32, no. 1, pp. 41–43, 2004.

[41] Tong, H., Prakash, B. A., Eliassi-Rad, T., Faloutsos, M., and Faloutsos, C., "Gelling, and melting, large graphs by edge manipulation," in *ACM CIKM*, 2012.

[42] Tong, H., Prakash, B. A., Tsourakakis, C., Eliassi-Rad, T., Faloutsos, C., and Chau, D. H., "On the vulnerability of large graphs," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pp. 1091–1096, IEEE, 2010.

[43] Wang, Y., Chakrabarti, D., Wang, C., and Faloutsos, C., "Epidemic spreading in real networks: An eigenvalue viewpoint," in *IEEE SRDS*, pp. 25–34, 2003.

[44] Xiao, R., Gong, X., and Yu, T., "Malware diffusion behavior analysis in the internet and its immune protection strategies," in *IEEE WAC*, 2012.

[45] Yannakakis, M., "Node-and edge-deletion np-complete problems," in *Proceedings of the tenth annual ACM symposium on Theory of computing*, pp. 253–264, ACM, 1978.

[46] Yannakakis, M., "Edge-deletion problems," *SIAM Journal on Computing*, vol. 10, no. 2, pp. 297–309, 1981.